Київський національний торговельно-економічний університет
Кафедра програмної інженерії та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка клієнт-серверної інформаційної системи розподілу навантаження викладачів кафедри»

Студента 2м курсу, 5 групи,
спеціальності
6.050126«Інженерія
програмного забезпечення»
спеціалізації
«Інженерія програмного
забезпечення»

_____
підпис студента

Івлєва Ігоря
Олександровича

Науковий керівник
науковий ступінь
вчене звання

_____
підпис керівника

Криворучко О.В.
д.т.н., професор

Гарант освітньої програми
науковий ступінь
вчене звання

_____
підпис керівника

Криворучко О.В.
д.т.н., професор

КИЇВ – 2019

					КНТЕУ 121– 05-13.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*	*Розробка клієнт-серверної інформаційної системи розподілу навантаження на викладачів кафедри*	*Стадія*	*Аркуш*	*Аркушів*
Зав. каф.		Криворучко О.В.				ТС	*1*	*68*
Керівник		Криворучко О.В						
Гарант		Криворучко О.В			*Титульна сторінка*	Факультет обліку, аудиту та інформаційних систем, 2м курс, 5 група		
Розроб		Івлєв І.О.						

ЗМІСТ

ВСТУП

Інформаційні технології набувають більшої актуальності і мають вплив на суспільний розвиток. Інформація стає одним з основних стратегічних ресурсів, а інформаційні технології - інструментом, з допомогою якого цей ресурс використовується. У світі освіти, ці технології до становлення відповідно до тенденцій розвитку інформаційної цивілізації. На теперішній час структурні підрозділи закладів вищої освіти активно використовують інформаційні технології для ефективної роботи працівників. До таких інформаційних технологій можна віднести: представницькі інтернет-джерела та автоматизовані системи управління університету та його структурного підрозділу. Управління структурними підрозділами це трудомісткий процес, що вимагає великих витрат ресурсів не тільки енергетичних, матеріальних, а й людських.

Наукова новизна полягає у використані нових технології розробки планування робіт викладачів кафедри, представлена як задача рівномірного наближення, що дозволяє на відміну від традиційних підходів забезпечити задані для викладачів обсяги робіт з урахуванням посадових коефіцієнтів і прийнятих при розподілі обмежень і сформувати навчальне навантаження відповідаючи сучасним рішенням.

Актуальність обраної теми полягає в тому, що практично у всіх катедрах розподіл роботи створює певну проблему. Слід зазначити, що головним чинником успішної продуктивності викладачів катедри є чисельність самих викладачів: так, якщо недостатня чисельність викладачів припускає до додаткового навантаження викладача, що може

привести до зменшення якості підготовки кваліфікованих спеціалістів і навпаки, якщо велика кількість науково-педагогічних співробітників то такий стан обумовлює до додаткових витрат закладів вищої освіти, що в умовах обмеженого фінансування бюджету призводить до скорочення витрат.

Особливість об'єкта управління, яким є освітня діяльність, і астенічний розвиток інформаційних систем для закладів освітніх послуг роблять актуальним завдання розробки автоматизованих систем управління ресурсами процесів життєвого циклу освітньої діяльності ЗВО та його основних підрозділів – катедр. На теперішній час існує велика кількість автоматизованих систем управління розподілу навантаження викладачів катедр, але в більшості вони не відповідають сучасним нормам та мають недосконалий функціонал. Основна мета даної роботи є вирішення проблематики даної теми шляхом розробки альтернативної автоматизованої системи управління розподілу навантаження на викладачів катедри за допомогою певних інструментів розробки програмних продуктів.

Наукова проблема даної теми полягає у оптимізації розподілу навантаження на викладачів катедри. Робоче навантаження вимірюється у кількостях контактних годин (запланованих годин підготовки, так і часу). Процес оптимізації ґрунтується за критеріями ранжування наукових і науково-педагогічних працівників, а також робочий час науково-педагогічних, наукових і педагогічних працівників, який закріплений і визначений у статті 56 ЗУ «Про вищу освіту».

Ціль даної роботи є розробка веб додатку розподілення навантаження на викладачів катедри, який має можливість розподілити навантаження між науково-педагогічними працівниками, вносити в систему зміни та зменшити витрати тимчасових і людських ресурсів на розподіл навантаження на катедрі.

Для досягнення цілей в рамках цієї випускної-кваліфікаційної роботи потрібно вирішити наступні питання:

- визначити вхідну і вихідну інформацію;
- визначити основні інструменти для розробки програмного продукту;
- визначити основні функції системи.

Об'єкт розробки – діяльність кафедри закладу вищої освіти.

Мета розробки – створення клієнт-серверної інформаційної системи з можливістю використання певного функціоналу.

Предмет дослідження – процес програмування клієнт-серверної інформаційної системи.

Метод проектування – мова програмування Node.js, використання фреймворку React.js, мова програмування JavaScript, система база даних MongoDB.

РОЗДІЛ 1
ОСНОВНІ ПОНЯТТЯ ПРО ОБЛІК МЕТОДИЧНОЇ, НАУКОВОЇ РОБОТИ ПЕДАГОГІЧНИХ І НАУКОВО-ПЕДАГОГІЧНИХ ПРАЦІВНИКІВ УНІВЕРСИТЕТУ

1.1. Поняття та принципи планування навчального навантаження

Багато проблем, що виникають у сфері освіти, успішно вирішуються за допомогою дискретних моделей і методів оптимізації. Це проблеми календарного планування, генерації тестів академічної оцінки успішності [8] і так далі. Науково-педагогічні працівники – це особи, які за основним місцем роботи у вищих навчальних закладах провадять навчальну, методичну, наукову (науково-технічну) та організаційну роботу.

Розподіл академічної навантаження (РАН) для викладачів є важливим етапом організації якісного навчального процесу. Важко формалізувати цю проблему. Це стає актуальним для великого і різного персоналу кафедри, досить багато курсів, включаючи наявність навантаження в декількох групах або на декількох факультетах, в разі змінною навантаження, коли існують особливі вимоги до розподілу академічного навантаження в університеті. Зокрема, роботи зосереджені на побудові математичних моделей для завдання РАН.

Проблема РАН, при якій середня кількість навчальних курсів, що призначаються кожному викладачеві мінімізується.

В результаті застосування моделі розподілу навантаження може бути отримано рішення, в якому одиниця навчального курсу розподіляється між кількома викладачами.

Це суперечить існуючій практиці в ЗВО. Двокрітеріальне формулювання проблеми РАН, в якій навчальні курси складаються з певних одиниць, кожна з яких може бути розподілена тільки одному викладачу. Мінімальна і максимально можливу кількість академічного навантаження призначається кожному викладачеві.

Планування всіх видів робіт науково-педагогічного працівника здійснюється відповідно до переліку основних видів діяльності та норм часу, що встановлені чинними нормативними документами. Річне навантаження науково-педагогічного працівника формується з таких основних видів діяльності як навчальна, методична, наукова, організаційна, виховна робота та інші види робіт.

Процес оптимізації ґрунтується за критеріями ранжування наукових і науково-педагогічних працівників, а також робочий час науково-педагогічних, наукових і педагогічних працівників, який закріплений і визначений у статті 56 ЗУ «Про вищу освіту». Згідно абзацу третього частини другої статті 56 визначає, що максимальне навчальне навантаження на одну ставку науково-педагогічного працівника не може перевищувати 600 годин на навчальний рік. Також в частині першій цього закону зазначено, що робочий час науково-педагогічних працівників становить 36 годин на тиждень. Слід зазначити, що у закладах вищої освіти мінімальний та максимальний обов'язковий обсяг навчального навантаження викладача в межах його робочого часу встановлює заклад вищої освіти з урахуванням виконання ним інших обов'язків (методичних, наукових, організаційних) і у порядку, передбаченому його статутом та колективним договором.

Обсяги різних видів робіт, які виконуються викладачами, встановлюються в залежності від контингенту, який навчається (студенти, аспіранти, докторанти, слухачі тощо), та необхідності долучення викладачів до різних форм навчальної, методичної, наукової та

організаційної роботи відповідно до їхніх індивідуальних можливостей та забезпечення найбільш ефективного використання творчого потенціалу. Залучення науково-педагогічних працівників до роботи, не обумовленої трудовим договором (контрактом), може здійснюватися лише за їхньою згодою або у випадках, передбачених законом.

1.2. Формування загального обсягу навантаження

Загальне навантаження кафедр формується на кожний навчальний рік на основі робочих навчальних планів за напрямами підготовки (спеціальностями) відповідно до планового контингенту студентів. Робочий навчальний план розробляється на кожний навчальний рік на основі навчального плану з метою конкретизації планування навчального процесу відповідно до особливостей підготовки фахівців певного освітньо-кваліфікаційного рівня, галузі знань, напряму підготовки (спеціальності) в розрізі різних етапів навчального процесу за роками та семестрами.

Відповідальність за розробку та реалізацію робочого навчального плану певного напряму/спеціальності покладається на випускову кафедру. До переліку дисциплін робочого навчального плану входять нормативні дисципліни, дисципліни за вибором навчального закладу, що є обов'язковими для вивчення студентами даного напряму підготовки (спеціальності), та дисципліни вільного вибору студентів, які студенти обирають із запропонованого переліку навчальних курсів. Зміст вибіркової частини освітньої програми розробляється випусковою кафедрою і, після схвалення методичною радою, затверджується вченою радою відповідного інституту[2]. Підставою для розгляду питання щодо включення вибіркових навчальних дисциплін (або блоку дисциплін спеціалізації) до варіативної частини робочого навчального плану є подання кафедрою, яка має забезпечувати викладання вказаної дисципліни/дисциплін, наступних документів:

- обґрунтування доцільності включення вибіркової дисципліни/дисциплін до програми підготовки фахівців певного напряму (спеціальності) та її взаємозв'язку з іншими курсами програми;
- інформації про навчально-методичне забезпечення навчальної дисципліни/дисциплін;
- інформації стосовно відповідності ліцензійним вимогам якісного складу науково-педагогічного персоналу, який буде забезпечувати викладання даного курсу/курсів;
- анотації навчальної дисципліни/дисциплін.

Рішення щодо включення до переліку курсів вільного вибору тієї чи іншої вибіркової дисципліни або блоку дисциплін, орієнтованих на формування певної спеціалізації в межах напряму підготовки або спеціальності, приймається вченою радою інституту.

Обсяг вибіркової навчальної дисципліни визначається відповідною кафедрою і має становити не менше ніж 2 кредити ЄКТС. Кількість вибіркових дисциплін у навчальному плані повинна визначатися обсягами навчального навантаження (кількістю годин/кредитів), відведеного освітньо-професійною програмою для вибіркової частини. На підставі робочих навчальних планів проводиться визначення видів навчального навантаження з кожної дисципліни та здійснюються розрахунки за відповідними нормативами для кожного виду навчальної роботи. Відповідно до затвердженого планового контингенту студентів встановлюються обсяги навчальної роботи для підготовки за певними напрямами/спеціальностями в розрізі кафедр університету. На підставі затверджених робочих навчальних планів та планової кількості ставок професорсько-викладацького складу проводиться розрахунок загального обсягу навчального навантаження науково-педагогічних працівників Університету. Розподіл обсягів навчального навантаження та розрахунок штатів професорсько-викладацького складу кафедр здійснюється

навчальним відділом Центру академічного розвитку та організації навчального процесу на основі вихідних даних для планування навчальної роботи[3]. Викладання загально-університетських дисциплін циклу гуманітарної та соціально-економічної підготовки і циклу природничо-наукової та фундаментальної підготовки планується одноменним кафедрам, якщо інститут (факультет) не має спеціалізованої кафедри.

1.3. Висновки до розділу 1

Отже, у цьому розділі було визначено основні засади, поняття та принципи планування навчального навантаження. Було детально розглянуто формування загального обсягу науково-методичних працівників університету.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Загальна характеристика об’єкту інформаційної системи

Об'єктом автоматизації є кафедра. Основна мета функціонування кафедри - задоволення потреб у навчанні або підвищенні кваліфікації з використанням нових освітніх технологій і якісного забезпечення навчального процесу.

Ефективна реалізація освітніх технологій підготовки фахівців можлива лише при створенні відповідних організаційних, кадрових і матеріальних умов.

У зв'язку з цим до основних завдань кафедри відносяться:

- проведення за всіма формами навчання лекцій, лабораторних, практичних, семінарських та інших видів навчальних занять, передбачених навчальними планами, на високому теоретичному та науковому рівні;
- керівництво навчальної, виробничої та переддипломної практики, курсовим і дипломним проектуванням, курсовими та дипломними проектами (роботами), а також самостійною роботою студентів по вивченню дисциплін кафедри;
- проведення курсових іспитів і заліків;
- задоволення потреб суспільства і держави у кваліфікованих фахівцях з вищою освітою та науково-педагогічних кадрах вищої кваліфікації;
- розробка, подання на затвердження в установленому порядку і впровадження навчальних планів і вузівських освітніх стандартів за напрямами і спеціальностями підготовки, освітніх стандартів з навчальних

дисциплін кафедри, а також підготовка висновків за навчальними програмами, складеними іншими кафедрами;

- організація і проведення фундаментальних і прикладних наукових досліджень, науково-технічних і дослідно-конструкторських робіт відповідно до затверджених планів, в тому числі з проблем освіти;
- керівництво науково-дослідною роботою студентів;
- обговорення закінчених науково-дослідних робіт, впровадження результатів цих робіт у виробництво; рекомендації для опублікування закінчених наукових робіт;
- проведення заходів з організації виховної роботи серед студентів;
- розгляд індивідуальних планів навчально-виховної, наукової, методичної та іншої роботи співробітників кафедри;
- вивчення, узагальнення та поширення досвіду роботи кращих викладачів;
- надання допомоги починаючим викладачам в оволодінні педагогічною майстерністю;
- розробка і здійснення заходів щодо використання при проведенні навчальних занять сучасних інформаційних і комп'ютерних технологій, аудіо та відеотехніки, і інших технічних засобів;
- підготовка науково-педагогічних кадрів, розгляд дисертацій, що подаються до захисту співробітниками кафедри або за дорученням ректорату і спеціалізованих учених рад факультету іншими претендентами;
- організація участі викладачів, співробітників, студентів, аспірантів і докторантів кафедри в конкурсах, конференціях, семінарах, симпозіумах і т.д .;

Розподіл навчального навантаження - це процес розподілу годин навчального навантаження кафедри по професорсько-викладацькому

складу. Навчально-організаційної документацією як правило займаються заступники з навчальної роботи і методисти. При розподілі навчального навантаження бере участь професорсько-викладацький склад кафедри[15].

Процес методичної роботи кафедри являє собою комплекс заходів, спрямованих на методичне забезпечення навчального процесу, підвищення педагогічної майстерності викладачів, вдосконалення аудиторної та самостійної роботи студентів, всіх форм і методів навчальної роботи у ЗВО. Основна мета методичної роботи - створення умов, що сприяють підвищенню ефективності та якості освітнього процесу.

Діяльність кафедри керується за допомогою нормативної документації інших підрозділів, а також міністерства освіти і науки України, міжнародними нормативними актами (в разі ведення освітньої діяльності на світовому ринку освітніх послуг)[2].

Процес науково-дослідницької роботи полягає в плануванні на всіх рівнях управління, в участі студентів, аспірантів і співробітників кафедри в конкурсах наукових робіт, виставках, наукових конференціях і симпозіумах, в проведенні наукових досліджень

Основним процесом з точки зору мети функціонування кафедри є «Навчальна робота». У ньому відбувається безпосередньо навчання студентів відповідно до програми вищої професійної освіти. Відповідальним за виконання процесу є завідувач кафедри[8].

Процес «Навчальна робота» складається з: організації навчального процесу (безпосереднє проведення лекцій, лабораторних та іншої аудиторної та самостійної роботи, навчальних та виробничих практик, поточного, підсумкового та залишкового контролю знань, і випускної кваліфікаційної роботи), і планування навчального процесу. Організація навчального процесу здійснюється на підставі Статуту КНТЕУ і інших документів, що регламентують внутрішній розпорядок. Крім того,

навчальний процес повинен відповідати навчально-організаційної документації. Сам процес складається з наступних підпроцесів:

- організація теоретичного курсу: лекції, практичні заняття (семінари), лабораторні роботи та інші види аудиторних занять, а також самостійна робота студентів, курсове проектування та ін .;
- організація проходження практик: керівництво навчальними і виробничими практиками, переддипломної практики;
- організація державного іспиту;
- організація проектування і захисту випускної кваліфікаційної роботи (керівництво дипломним проектуванням, нормоконтроль, і ін.);
- організація поточного (лабораторні та контрольні роботи, колоквіуми, і ін.), Підсумкового (курсові проекти, залікові та екзаменаційні одиниці) і залишкового контролю знань.

Таким чином, модель «як є» в цілому збігається з моделлю «як повинно бути», однак можлива поява додаткових подпроцессов, пов'язаних з введенням первинної інформації та її автоматизованою обробкою.

2.2. Постановка завдання при проектуванні програмного забезпечення

Враховуючи загальну діяльність об’єкту інформаційної системи можна сформулювати постановку завдань, а саме реалізувати АСУ розподілу навантаження викладачів кафедри. АСУ повинна забезпечувати підтримку режиму групової роботи користувачів за для ефективної роботи кафедри. АСУ повинна бути побудована на системі клієнт-сервер (далі - Web-додаток) з доступом до неї за допомогою мережі Інтернет.

Основні функції веб-додатку можна віднести: 1) авторизація користувачів 2) функція адміністрування системи 3) функції користувачів. Детальніше представлені основні функції у веб-додатку, рисунок 2.1.

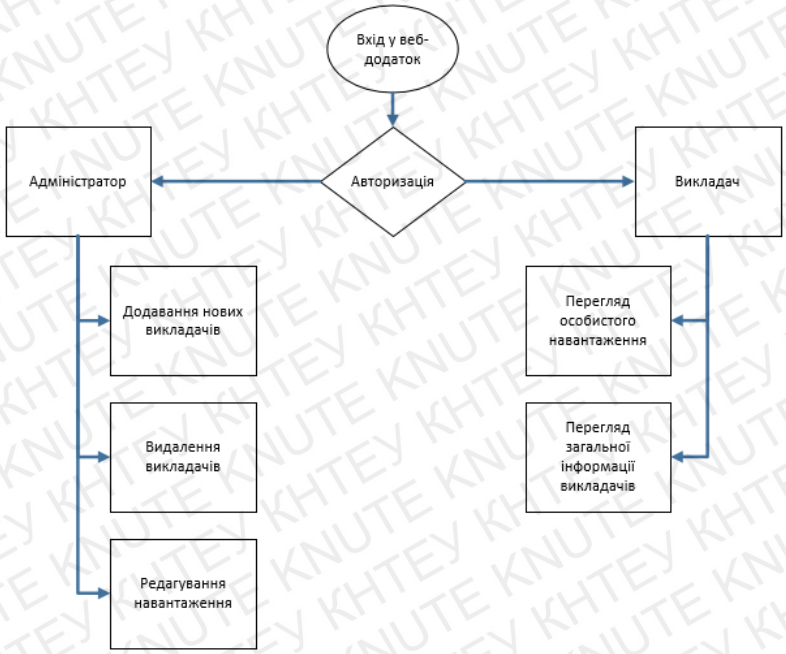

Рис. 2.1. Блок-схема функцій у веб-додатку

Веб-додаток повинен мати інтерфейс для авторизації з полями вводу даних такі як e-mail та пароль. Дані всіх облікових записів зберігаються в базі даних. Під час входу веб-додаток перевіряє цілісність та валідність даних. Веб-додаток має частину адміністратора з певними функціями:

Додавання нових викладачів.

За допомогою інтерфейса користувача додавання новини Веб-додаток повинен надати адміністратору інтерфейс додавання нових викладачів, а саме посилати на сторінку додавання викладачів, в якій містяться поля для введення особистих даних викладачів. Також серверна та логічна частина веб-додатку повинна перевіряти правильність введених даних.

Видалення викладачів.

Веб-додаток повинен надавати адміністратору можливість видалення викладачів за допомогою функціональних кнопок. Також серверна частина повинна надіслати запит на видалення викладача з бази даних.

Редагування навантаження.

Адміністратор веб-додатку має можливість змінювати навантаження на кожного викладача або видаляти навантаження. Навантаження на

викладач повинна мати форму таблиці з даними про назву предмета, кількість лекцій, кількість екзаменів та заліків.

Викладачі мають можливість перевіряти особисте навантаження та ознайомитись з загальною інформацією з викладачами кафедри.

Не функціональними вимогами є:

- Зручність використання − інтерфейс користувача повинен бути зрозумілий і легкий у використанні.
- Надійність − система повинна бути в працездатному стані двадцять чотири години на добу сім днів на тиждень, час простою − нуль процентів.
- Продуктивність − система повинна надавати можливість її покращення
- Безпека − система повинна дозволяти користувачам корегувати інформацію лише в межах своєї компетентності шляхом встановлення паролів та логіну.
- Мати можливість додати новий функціонал без зміни високорівневої архітектури
- Коректо працювати в web-браузерах: IE8 +, Firefox 3.5+, Opera 10.5+, Google Chrome 6 +, Apple Safari 3+
- Підтримка кодування UTF-8.

Для розробки цього проекту потрібно використовувати наступні системні вимоги:

- операційну систему Linux, MacOS або Windows;
- сервер Node.js, фреймворк React.js, мову розмітки HTML.
- сервер баз даних MongoDB.

Також до проектування веб-додатку потрібно створювати сценарії функціональних дій адміністратора та викладачів. На рисунку 2.2. зображено сценарій можливостей адміністратора.

Рисунок 2.2. Сценарій можливостей адміністратора.

2.3. Огляд сучасних програмних забезпечень розподілу навантаження на викладачів кафедри

На теперішній час існує велика кількість програм для автоматизації розподілу навантаження викладачів кафедр університету. Згідно інформації наданої з мережі Інтернет можна зробити висновок, що більшість університетів використовую свою інформаційну систему, призначену для розподілу навантаження на кафедрі. Цей трудомісткий процес дуже складно описати математично з огляду на те, що потрібно враховувати дуже велику кількість входжень інформації та умов[17]. До таких належить професорсько-викладацький склад, який може змінюватися в часі і, згідно з яким, пред'являються вимоги до розподілу основного навантаження. Такі питання вирішити дуже складно, так як потрібно враховувати індивідуальність кожного викладача і ймовірно його статус на кафедрі, що передбачає розробку ускладненою

комплексної системи або використання експертної складової (в даному випадку завідувача кафедри)[19].

Також існують програми розроблені приватними компаніями. Серед приватних додатків розподілу навантаження є:

Автоматизована система «Навантаження ВНЗ».

Вона забезпечує комплексний підхід до формування та розподілу навчального навантаження установ ВПО. Система розрахована для роботи в локальній мережі і має три рівні доступу, які визначають функціонал доступний користувачам.

Адміністратору системи доступні наступні функції:

- перевірка навчальних планів на спадкоємність;
- формування відомостей про очікуване контингенті студентів;
- створення списку навчальних груп на основі контингенту студентів;
- завдання норм на прийом заліків та іспитів, керівництво дипломними, курсовими, дисертаційними та іншими видами робіт;
- централізоване перейменування дисциплін та закріплення їх за кафедрами;
- визначення параметрів формування потоків і навчального навантаження;
- формування навчального навантаження кафедр на базі навчальних планів і списку груп;
- розрахунок штатного розкладу кафедр;

Завідувачам кафедр доступні наступні функції:

- закріплювати навчальне навантаження за викладачем або передавати її іншій кафедрі;
- об'єднувати групи в потоки і розбивати групу на підгрупи з різних видів занять, наприклад, лабораторним, практичним або курсових робіт;

- враховувати дані про розподіл навчального навантаження попереднього навчального року;
- уточнювати навантаження протягом навчального року;
- створювати доручення кафедр, що експортуються в програму автоматичного упорядкування розкладу занять;
- виконувати перевірку коректності та відповідності розподіленого навантаження даними навчального відділу.

Викладачам доступні наступні функції:

- одержувати розгорнуту інформацію по навчальному навантаженні;
- заповнювати необхідну кількість годин на підготовку до занять;
- формувати свої індивідуальні плани, в яких можна конкретизувати зміст робіт, виконуваних у «другій половині дня» (наукова, організаційно-методична, навчально-методичну, виховна робота);
- вести облік побажань викладачів за графіком роботи протягом тижня;

«БІТ.ЗВО.ОБЛІК НАВАНТАЖЕННЯ ВИКЛАДАЧІВ»

Програма дозволить автоматизувати планування і облік виконання навантаження професорсько-викладацького складу. Данна програма дозволяє вирішувати такі завдання:

- перевести за допомогою нормативів дані, отримані від факультетів з фізичної форми в годинну навантаження і розрахувати загальну річну навантаження кафедри, диференційовану за видами навантажень;
- розподілити цю навантаження між викладачами кафедри, беручи до уваги не тільки рівномірність розподілу, а й індивідуальні особливості кожного викладача і його роль в житті кафедри, тобто, враховуючи, працює викладач на повну ставку, або на частину її,

навантажений він наукової або адміністративної роботою, веде додаткові заняття, пише підручник або монографію та ін .;

- скласти для кожного викладача зведену таблицю, де зібрані всі навчальні навантаження, в яких він бере участь, розбиті по семестрах і формам навчання (очна і заочна);
- підготувати первинний матеріал для складання розкладу занять кафедри (дані про кількість груп, розподілених кожному викладачеві в 1 і 2 семестрах з урахуванням форми навчання, дані про його тижневої годинний навантаженні та ін.).

Кафедра «Програмної інженерії та кіберзбезпеки» забезпечує розподіл навантаження на кафедрі за допомогою додатку, створеним на базі Microsoft Excel з використанням макросів. Основні переваги використання Microsoft Excel такі:

- Додавати нових викладачів, в першому варіанті в список
- Користування програмою за рахунок зв'язування файлів
- Друкування звітів
- Вихідні документи зберігаються кожен в окремому документі

Недоліками використання є:

1. Немає можливості запам'ятовувати викладачів, і тимчасово виключати їх з навчального плану. Наприклад, викладач взяв лікарняний на досить тривалий термін, і не може викладати семестр або навіть протягом року, тоді його потрібно виключити з навчального плану, але запам'ятати в професорсько-викладацький склад.

2. Для того щоб подивитися різні звіти необхідно відкривати новий документ. Що призводить до втрати робочого часу лише за рахунок звернення до додатка операційної системи Windows - explorer.exe.

3. Під час перегляду основного звіту видно не всю сторінку, яка має досить великий обсяг, а, отже, обмеженість перегляду інформації

викликає дискомфорт і знову ж витрачається робочий час на прокрутку сторінки.

4. При розрахунку ставок не враховується, що на комерції і на бюджеті вчиться різну кількість студентів в різних групах, різні види навантажень має різну трудомісткість, а викладачі мають ступеня, звання та ін. Це окремий розрахунок, який вимагає додаткової обробки.

2.4. Огляд програмних засобів для розробки програмного забезпечення

Згідно поставленого завдання програма розподілу навантаження викладачів є клієнт-серверним додатком і доцільно використовували веб-технології для реалізації програмного забезпечення. На теперішний час існує різноманітні технологій розробки веб-додатків. Будь-який веб-додаток зроблений з використанням декількох технологій. Комбінації цих технологій називаються «стеком», популяризується стеком LAMP, який є абревіатурою для Linux, Apache, MySQL, PHP, всіх компонентів з відкритим вихідним кодом. У міру розвитку веб-розробки і їх інтерактивності на перший план виходили односторінкові додатки (SPA). SPA - це парадигма веб-додатків, в якій відсутнє оновлення веб-сторінки для відображення нового вмісту, замість цього використовуються спрощені звернення до сервера для отримання деяких даних або фрагментів і оновлення веб-сторінки. Результат виглядає досить витончено в порівнянні зі старим способом повного перезавантаження сторінки[21].

Стек MERN - це JavaScript-стек, розроблений для спрощення процесу розробки. MERN включає в себе чотири компоненти з відкритим вихідним кодом: MongoDB, Express, React і Node.js. Серед цих технологій Mongo DB - це система баз даних, Node JS - внутрішнє середовище виконання, Express JS - внутрішня веб-інфраструктура, а React – інтерфейсне середовище та Redux – інструмент управління станом.

Express.js

Express - це фреймворк для веб-додатків, в якому виконується код JavaScript-додатку. Express працює як модуль в середовищі Node.js. Express може обробляти маршрутизацію запитів до потрібних частин вашої програми (або до різних програм, які працюють в одному середовищі). Express надає можливість згенерувати остаточний HTML-код, який буде відображатися браузером користувача. Також Express можна використовувати для простого надання REST API - надання зовнішньому додатком доступу до ресурсів, які йому необхідні, наприклад, до бази даних.

React.js

React - це бібліотека JavaScript, яка використовується для створення користувацьких інтерфейсів. React використовується для розробки односторінкових додатків і мобільних додатків завдяки своїй здатності обробляти швидко мінливі дані. React дозволяє користувачам кодувати в JavasScript і створювати компоненти користувальницького інтерфейсу.

Бібліотека React може використовуватися для створення образів, що відображаються в HTML. React образи носять декларативний характер. Це означає, що розробникам не потрібно турбуватися про управління наслідками змін стану уявлення (об'єкта, що визначає поведінку компонентів) або змін до даних. React використовує повнофункціональну мову програмування (JavaScript) для створення повторюваних або умовних елементів DOM. З React один і той же код може виконуватися як на сервері, так і в браузері.

Redux

Redux - це інструмент управління як станом даних, так і станом інтерфейсу в JavaScript-додатках. Він підходить для односторінкових додатків, в яких управління станом може з часом стає складним.

MongoDB

MongoDB - це NoSQL (нереляційних) документно-орієнтована база даних. У той час як традиційні реляційні бази даних мають типовий дизайн схеми, заснований на стовпчиках і таблицях, MongoDB не вимагає схем. Дані зберігаються в гнучких документах за допомогою мови запитів на основі JSON (JavaScript Object Notation). Зміст, розмір і кількість полів в документах можуть відрізнятися від одного до іншого. Це означає, що структура даних буде змінюватися з часом.

Переваги вибору стека MERN:

- Охоплює весь цикл веб-розробки від розробки на стороні клієнта до розробки на стороні сервера використанням JavaScript.
- Підтримує архітектуру MVC (Model View Controller) для забезпечення плавності процесу розробки.
- За допомогою стека JavaScript розробники повинні володіти тільки JavaScript і JSON.
- Можливість реалізації чотирьох кращих технологій: MongoDB, ExpressJS, React і NodeJS.
- Поставляється з готовим великим набором інструментів тестування.
- Відкритий вихідний код.

2.5. Висновки до розділу 2

Отже, у цьому розділі була визначена загальна характеристика об’єкту інформаційної системи. Була визначена постановка завдання при проектування програмного забезпечення. Також був проведений огляд сучасних програмних забезпечень розподілу навантаження на викладачів кафедри та системний аналіз всіх застосованих технологій для розробки веб-додатку.

РОЗДІЛ 3
АНАЛІЗ АРХІТЕКТУРИ ІНФОРМАЦІЙНОЇ СИТЕМИ

3.1. Загальна характеристика об’єкту інформаційної системи

Веб-додаток має певну кількість складових частин, які взаємодіють між собою і представляється як ціле програмне забезпечення. Веб-додаток розподілу навантаження викладачів кафедр складається з таких частин як:

- Клієнтська частина
- Логічна частина відображення UI компонентів
- Джерело даних
- Маршрутизація обробки запитів

Для програмування всіх структурних частин веб-додатку потрібно встановити взаємозв’язок між ними. В першу чергу взаємозв’язок відображається між такими складовими частинами:

Клієнтська частина:

Клієнтська частина передбачає роботу веб-додатку в браузері користувача. для оптимального задоволення більшості вимог було прийнято рішення розробити веб-додаток з використанням технології Single Page Application.

SPA - концепція односторінкового веб-додатку, що припускає використання архітектурного рішення "Товстий клієнт" (надає собою функціональний інтерфейс користувача з виконанням бізнес-логіки) і API серверної частини для обміну даними [7].

Особливість "Товстого клієнта" в даному випадку полягає в тому, що практично всі проміжні дії для отримання документа відбуваються на

клієнті і тільки дані принципових дій відправляються на сервер, де в якості відповіді приходять необхідні дані для досягнення користувальницької мети. Таким чином, це зменшує залежність від якості зв'язку між клієнтом і сервером. Без звернення клієнта на сервер, зміна уявлень інтерфейс користувача змінюється практично миттєво згідно UX [16]. У разі звернення, обмін даними відбувається в асинхронному режимі, що дозволяє продовжувати роботу користувача без блокування клієнта в очікуванні відповіді сервера. За рахунок винесення частини бізнес-логіки й обробки подань на клієнта, а також відділення уявлення від даних, - досягається слабка зв'язаність клієнтської і серверної частини веб-додатку.

Таким чином, в порівнянні з традиційним підходом організації ВП, робота клієнтської частини ВП характеризується високою швидкодією (проте має залежність від апаратних характеристик клієнта) і можливістю коректної роботи при неякісному з'єднанні.

Клієнтська частина веб-датку являє односторінкову архітектуру з різними представленнями такі як:

Авторизація:

Сторінка авторизації є головною сторінкою і виконує функцію входу користувача в веб-додаток. Для входу в веб-додаток користувач повинен ввести в необхідні поля особистий email та пароль. Згідно з введеними даними визначається роль користувача і його підрозділ, якщо його роль не «Простий користувач».

Загальне навантаження:

Сторінка «Загальне навантаження» є головною сторінку у адміністратора веб-додатку. В ній міститься дані про загальне навантаження на викладачів кафедри. Також міститься функціонал редагування, додавання і видалення навантаження на викладачів.

Зведене навантаження:

Сторінка «Зведене навантаження» являє собою список викладачів з їхнім навантаженням. Функціональна частина цієї сторінки є можливість перегляду навантаження певного викладача.

Викладачі:

Сторінка «Викладачі» являє собою впорядкований список викладачів кафедри в якій вказано прізвище та ініціали, науковий ступінь та контактні дані викладача. Вхідними даними є дані з бази даних.

Обліковий запис:

Сторінка «Обліковий запис» являє собою перегляд особистих даних авторизованого користувача. Вхідними даними є дані з бази даних.

Додати викладача:

Сторінка «Додати викладача» є сторінкою з доступом адміністратора веб-додатку. Вона дає можливість додавати нового викладача у веб-додаток та записувати нові облікові дані та навантаження у базу даних.

Логічною частиною відображення UI компонентів являє собою JavaScript файл в якому міститься клас в якому є:

- Функція авторизації та деавторизації;
- Функція відображення загального навантаження на викладачів кафедри;
- Функція відображення зведеного навантаження на певного викладача кафедри;
- Функція редагування, видалення та додавання навантаження на певного викладача кафедри;
- Функція додавання нового викладача кафедри;
- Функція валідації введених даних.

Джерело даних.

Дані веб-додатку необхідно систематизовано зберігати і обробляти, як чого використовується база даних. На підставі передумов і вимог, було зроблено висновок, що необхідна СУБД, яка пропонує:

- Підтримку NoSQL.
- Можливість горизонтальної масштабованості, зокрема реплікації БД

Також перевагою буде підтримка даних типу JSON, так як JSON обраний в якості формату обміну даними між клієнтом і API. А також поширеність і документація. Згідно для оптимального рішення поставлених завдань була обрана СУБД MongoDB.

MongoDB - документно-орієнтована СУБД [11]. Зберігає дані в форматі BSON, що являє собою бінарну форму подання JSON. Таким чином, запис в MongoDB є аналогом об'єктного уявлення у форматі JSON. MongoDB має інтерфейс для виконання операцій на мові JavaScript, можливості асинхронної реплікації, підтримує необхідні типи даних, які можуть знадобитися при проектуванні БД згідно предметної області, наприклад, масив, дата, об'єкт і інші. Раціонально підходить для зберігання ієрархічних даних [12].

Маршрутизація обробки запитів.

Архітектура серверної частини побудована на основі фреймворку Express. Він інкапсулює обробку високорівневих об'єктів додатки, наприклад, такі як об'єкти запиту і відповіді, надаючи їх в розпорядження розробнику[21].

Для структурування серверної частини ВП були взяті принципи архітектурного типового рішення MVC. Розглянемо основні компонентні уявлення серверної частини ВП, які були розроблені:

- Маршрутизатор (Route) - відповідно до маршруту делегує подальшу обробку запиту керуючим об'єктів нижчого рівня: контролеру або API.
- Контролер (Controller) - відповідає за обробку запиту, визначає HTML для користувацького інтерфейсу. Містить методи для реалізації

функціональних вимог до ВП. Використовує модель для отримання даних предметної області.

- Модель (Model) - відповідає за бізнес-логіку безпосередньо пов'язану з предметної областю. Є моделлю суті. Надає інтерфейс для роботи з сутностями. Інкапсулює обробку даних відповідної їй суті. **Mongoose** – являє "Обгортку" для моделей, надає функціональність для роботи з моделями.

- Шаблон (Template) - уявлення користувацького інтерфейсу у вигляді HTML розмітки.

- Вид (View) - уявлення, код якого необхідно обробити сервером додатка.

- API - стандартизований інтерфейс для роботи з даними предметної області. Інкапсуляція бізнес-логіки за певним маршрутом. Повертає всі дані в форматі JSON. Запит до API визначається параметром "api" в рядку URL. Відповідно до REST, на прикладі роботи з документами, семантика запитів була побудована таким чином, таблиця 3.1.

Таблиця 3.1.

Семантика запитів відповідно до REST

Метод	**Доступ**	**GET**	**POST**	**PUT**	**DELETE**
api/profile	Private	Повернути профіль вибраного користувача	-	-	Видалити користувача та профіль
api/profile/all	Private	Повернути профілі всіх користувачів	-	-	-
api/profile/handle/:handle	Private	Повернути облікові дані користувача	-	-	-
api/profile/experience	Private	-	Додати навантаження на користувача	Змінити навантаження користувача	
api/users/register	Private	-	Реєстрація нового користувача	-	-
api/users/login	Public	Авторизація користувача	-	-	-
api/users/current	Private	Повернути дані користувача	-	-	-

В основі серверної частини веб-додатку міститься:

- Компоненти, які залежать від коректної роботи веб-додатку;
- Маршрутизатор між моделями адміністратора та користувачами;
- Конфігуратор авторизації користувачів та адміністратора;.
- Підключення до бази даних
- Підключення до статичних медіяфайлів;
- Конфігуратор сесії адміністратора та користувача
- Маршрутизатор між шаблонами;
- Створення локального сервера.

Також згідно вимог розроблена уявлення користувацького інтерфейсу які розташовуються у відповідних папках. У папці «client» зберігатися компоненти написані мовою JavaScript. До відокремлення користувацького інтерфейсу використовувати метод розподілу компонентів між інтерфейсом користувача та адміністратора. В таблиці 3.2. представлено компоненти інтерфейсу адміністратора[15]. В таблиці 3.3. представлено компоненти інтерфейсу користувача.

Таблиця 3.2.

Компоненти інтерфейсу адміністратора.

Компонент	**Опис**
MainTable	Загальне навантаження
Profiles	Навантаження викладачів
AddPersonalTable	Додати навантаження
AddTeacher	Додати викладача
MyProfile	Профіль користувача
Teachers	Список користувачів

Таблиця 3.3.

Компоненти інтерфейсу користувача.

Компонент	**Опис**
Teachers	Список користувачів
MyProfile	Профіль користувача
TableData	Навантаження викладача

Технологія обробки запитів

Основним процесом роботи веб-додатку – це обмін і обробка даних. В сучасних потребах доцільно розроблювати швидкі методи для обміну даних. **AJAX** - це абревіатура, що розшифровується як Asynchronous JavaScript and XML, і описує набір методів розробки, що використовуються для створення веб-сайтів та веб-додатків. Запити користувачів обробляються дуже швидко, тому що використання технології AJAX дозволяє не перезавантажувати сторінку цілком, а оновлювати на ній тільки ті елементи, які потребують оновлення. AJAX - це просто абревіатура, що позначає підхід до створення веб-додатків за допомогою наступних технологій:

- AJAX використовує XHTML для вмісту, CSS для презентації, а також Модель об'єкта документа та JavaScript для динамічного відображення вмісту.
- Звичайні веб-додатки передають інформацію до та від серйозних, використовуючи синхронні запити. Це означає, що ви заповнюєте форму, натискаєте надіслати та направляєте на нову сторінку з новою інформацією від сервера.
- З AJAX, коли ви натиснете надіслати, JavaScript подасть запит на сервер, інтерпретує результати та оновить поточний екран. У чистому сенсі користувач ніколи не дізнається, що щось навіть передається на сервер.

• XML зазвичай використовується як формат для отримання даних сервера, хоча будь-який формат, включаючи звичайний текст, може бути використаний.

• AJAX - це технологія веб-браузера, незалежна від програмного забезпечення веб-сервера.

• Користувач може продовжувати користуватися програмою, поки клієнтська програма вимагає інформацію від сервера у фоновому режимі.

• Інтуїтивна та природна взаємодія користувача. Клацання не потрібно, рух миші є достатнім тригером події.

Процедура роботи AJAX відрізняється ніж звичайна модель. Порівняльна характеристика детально представлено в рисунку 3.1.

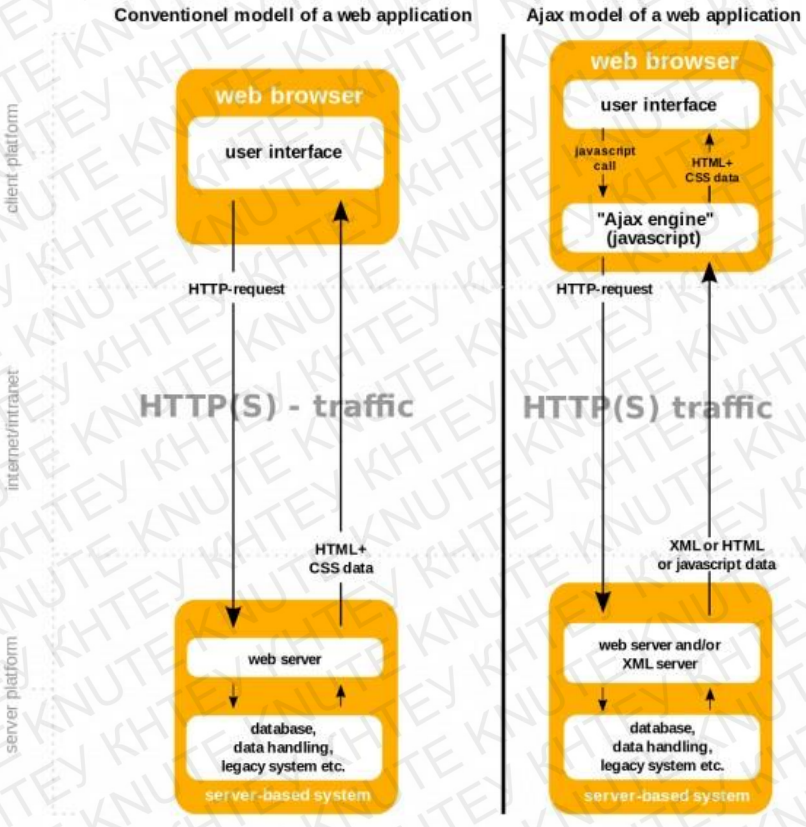

Рисунок 3.1. Діаграма роботи звичайної моделі та AJAX моделі.

Звичайна модель:

• Запит HTTP надсилається з веб-браузера на сервер.

- Сервер отримує та згодом отримує дані.
- Сервер надсилає запитувані дані у веб-браузер.
- Веб-браузер отримує дані та перезавантажує сторінку, щоб дані з’являлися.
- Під час цього процесу користувачі не мають іншого вибору, як чекати, поки весь процес буде виконаний. Це не тільки забирає багато часу, але й покладе зайве навантаження на сервер.

Модель AJAX:

- Браузер створює виклик JavaScript, який потім активує XMLHttpRequest.
- На задньому плані веб-браузер створює HTTP-запит до сервера.
- Сервер отримує, завантажує та відправляє дані назад у веб-браузер.
- Веб-браузер отримує запитувані дані, які будуть безпосередньо відображатися на сторінці. Перезавантаження не потрібно.

3.2. Побудова структури програмного забезпечення

Побудова файлової структури проекту дозволить розподілити програмний код за його функціональним призначенням. Потрібно визначити першочергові точки програмування з яких розпочнеться побудова програмного рівня веб-додатку[21].

При побудові файлової структури проекту слід врахувати функціональний підхід, тобто створити файли які будуть відповідати за збереження окремих частин коду і їх підключення безпосередньо у процесі програмування.

Це є важливим для створення великих веб-додатків. На рисунку 3.2. зображено структуру файлів проектованого веб-додатку.

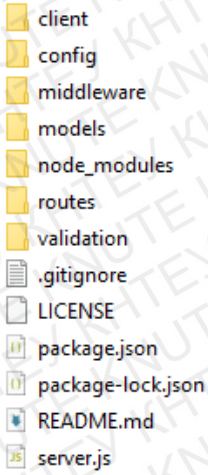

Рисунок 3.2. Структура файлів проектованого веб-додатку

Функціонування роботи веб-додатку залежить від цілої сукупності взаємозв'язаних файлів. Клієнтська частина веб-додатку зберігається у папці «client». В цій папці зберігається всі шаблони які відображаються користувачу та адміністратору веб-додатку.

Основна серверна частина веб-додатку зберігається у файлі «server.js» та у папках «routes», «validation», «config» та «middleware».

У цих файлах зберігаються всі методи та функції вищезазначених даних.

Схеми бази даних зберігаються у папці «models» в якій зберігаються схеми профілю користувача та користувачі.

3.3. Структура бази даних програмного забезпечення

На основі аналізу предметної області і вимог, що пред'являються до підсистеми розподілу навчального навантаження, була побудована наступна логічна модель.

Однією з основних частин інформаційного забезпечення є інформаційна база. Інформаційна база (ІБ) являє собою сукупність даних, організована певним способом і збережена в пам'яті обчислювальної системи у вигляді файлів, за допомогою яких задовольняються інформаційні потреби управлінських процесів і вирішуваних завдань. Розробка БД виконується за допомогою моделювання даних. Мета

моделювання даних полягає в забезпеченні розробника ІС концептуальною схемою бази даних у формі однієї моделі або кількох локальних моделей, які відносно легко можуть бути відображені в будь-яку систему баз даних. Найбільш поширеним засобом моделювання даних є діаграми «сутність-зв'язок» (ERD).

Кожна сутність повинна мати унікальний ідентифікатор. Кожен екземпляр сутності повинен однозначно ідентифікуватися і відрізнятися від всіх інших примірників даного типу сутності. Кожна сутність повинна володіти деякими властивостями:

- Мати унікальне ім'я; до одного і того ж імені повинна завжди застосовуватися одна й та ж інтерпретація; одна і та ж інтерпретація не може застосовуватися до різних іменах, якщо тільки вони не є псевдонімами;
- Мати один або кілька атрибутів, які або належать сутності, або успадковуються через зв'язок;
- Мати один або кілька атрибутів, які однозначно ідентифікують кожен екземпляр сутності.

Кожна сутність може мати будь-якою кількістю зв'язків з іншими сутностями моделі. Логічний рівень - це абстрактний погляд на дані, коли дані представляються так, як виглядають в реальному світі, і можуть називатися так, як вони називаються в реальному світі, наприклад «Прізвище співробітника». Об'єкти моделі, що представляються на логічному рівні, називаються сутностями і атрибутами. Логічна модель даних може бути побудована на основі іншої логічної моделі, наприклад на основі моделі процесів. Логічна модель даних є універсальною і ніяк не пов'язана з конкретною реалізацією СУБД [15].

Фізична модель даних, навпаки, залежить від конкретної СУБД, фактично будучи відображенням системного каталогу. У фізичної моделі міститься інформація про всі об'єкти БД. Оскільки стандартів на об'єкти

БД не існує (наприклад, немає стандарту на типи даних), фізична модель залежить від конкретної реалізації СУБД. Отже, однією і тією ж логічної моделі можуть відповідати кілька різних фізичних моделей. Якщо в логічної моделі не має значення, який саме тип даних має атрибут, то у фізичній моделі важливо описати всю інформацію про конкретних фізичних об'єктах - таблицях, колонках, індексах, процедурах і т.д.

Далі розглянемо структуру таблиць, які необхідно буде створити.

Таблиця 3.4.

Структура таблиці користувача

Поле	Тип	Унікальність	Обов'язковість	Назва	Примітка
Email	string	+	+	Електронна адреса	
Name	string	-	+	Ім’я	
Password	string	-	+	Пароль	
Avatar	string	-	-	Облікове зображення	
role	string	-	+	Роль користувача	«user» або «admin»

Таблиця 3.5.

Структура таблиці навантаження

Поле	Тип	Унікальність	Обов'язковість	Назва	Примітка
User	string	+	+	користувач	
Degree	string	-	+	ступінь	
Telephone	string	-	-	телефон	
Email	string	+	+	email	
Facebook	string	+	-	соц.мережі	
halfYear	number	-	+	півріччя	
trainningForm	number	-	+	форма навч	
faculty	number	-	+	факультет	
disciplinesName	string	-	+	назв.дисц.	
term	number	-	+	термін	
course	number	-	+	курс	
groupNumber	number	-	+	№.групи	
secondTeacher	string	-	-	викладач2	
lectionsNumb	number	-	+	к-сть лекцій	години
labsNumb	number	-	+	к-сть лаб.	години
consultaionsNumb	number	-	+	к-сть конс.	години
practicalNumb	number	-	+	к-сть практ.	години
ModularContNumb	number	-	+	к-сть мод.	години
ExamsNumb	number	-	+	к-сть екз.	години

3.4. Висновки до розділу 3

Отже, у цьому розділі була розглянута структура роботи програмного забезпечення. Побудована структура програмного забезпечення з вказаними директоріями. Також детально визначена структура бази даних програмного забезпечення.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Розробка серверної частини веб-додатку

Серверна частина веб-додатку відповідає за обробку та збереження даних в базі даних та відповідає за валідацію введених даних. Серверна частина веб-додатку розроблена на базі Node.js. Для досягнення гнучкої і багатої функціональності серверної частини на базі Node.js, доцільно використовувати фреймворк. Як приклад кращого Node.js фреймворку доцільно використовувати Node.js -фреймворк Express.js. Express - це легка структура веб-додатків, яка допомагає організувати веб-додаток в архітектурі MVC на стороні сервера.

В основі серверної частини веб-додатку міститься:

- Створення локального серверу;
- Підключення до бази даних;
- Побудова моделі колекцій та схеми бази даних;
- Процес авторизації користувача;
- Створення API.

Створення локального серверу.

Node.js підтримує можливість створення серверу по протоколу HTTP. Для того щоб створити сервер потрібно підключити модуль Express.

```
const express = require('express');
```

Для використання Express на початку треба створити об'єкт, який буде представляти додаток.

```
const app = express();
```

Для обробки запитів в Express визначено ряд вбудованих функцій, і однією з таких є функція app.get (). Вона обробляє GET-запити протоколу HTTP і дозволяє зв'язати маршрути з певними обробниками. Для цього першим параметром передається маршрут, а другим - обробник, який буде викликатися, якщо запит до сервера відповідає даному маршруту:

```
app.get('/', (req, res) => res.send(`api`))
```

Далі потрібно перевірити стан веб-додатку та встановити правильний порт серверу.

```
// Server static assets if in production
if (process.env.NODE_ENV === 'production') {
  // Set static folder
  app.use(express.static('client/build'));

  app.get('*', (req, res) => {
    res.sendFile(path.resolve(__dirname, 'client', 'build', 'index.html'));
  });
}
```

Завершальний етап створення серверу це запуск сервера. Для запуску сервера викликається метод app.listen (), в який передається номер порту.

```
const port = process.env.PORT || 5000;
app.listen(port, () => console.log(`Server running on port ${port}`));
```

Підключення до бази даних.

Базою даних є MongoDB. Для ефективної роботи з БД доцільно використовувати бібліотеку Mongoose. Mongoose представляє спеціальну ODM-бібліотеку (Object Data Modelling) для роботи з MongoDB, яка дозволяє зіставляти об'єкти класів і документи колекцій з бази даних. Щоб підключити БД до веб-додатку потрібно підключити mongoose

```
const mongoose = require('mongoose');
```

Далі потрібно під’єднати бібліотеку mongoose до БД, для цього є спеціальний метод mongoose.connect (), в який передається адреса бази даних на сервері mongo

```
// Connect to MongoDB
mongoose
```

```
  .connect(db, {
    useCreateIndex: true,
    useNewUrlParser: true
  })
  .then(() => console.log('MongoDB Connected'))
  .catch(err => console.log(err));
```

Параметр «db» це змінна в якій записані ключі доступу до виділеного сервера з базою даних MongoDB Atlas - це глобальна послуга хмарних баз даних для сучасних додатків.

Побудова моделі колекції та схеми бази даних.

Цілісність зв'язків в MongoDB не підтримується, а вирішується на рівні коду сервера веб-додатків. База даних веб-додатку має дві колекції: 1) колекція користувачів; 2) колекція навантаження та публічних даних користувача. Для того щоб мати уявлення відносин між колекціями, розроблена схема БД, на якій відображені псевдо-зв'язки. Псевдо-зв’язки між колекціями бази даних веб-додатку розподілу навантаження на викладачів кафедр зображено в додатку А.

Дані, які використовуються в Mongoose, описуються певною схемою. Схема містить метадані об'єктів. Зокрема, тут встановлюємо, які властивості матиме об'єкт і який у них буде тип даних. Веб-додаток має дві моделі колекцій даних:

Модель Користувача:

```
// User model
const mongoose = require('mongoose');
const UserSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true
  },
  avatar: {
    type: String
```

```
  },
  role: {
    type: String,
    default: "user",
    enum: ["user", "admin"]
  },
  date: {
    type: Date,
    default: Date.now
  }
});
module.exports = User = mongoose.model('users', UserSchema);
```

Модель профілю представлена в додатку Б.

Метод mongoose.model вказує на назву моделі. Mongoose потім буде автоматично шукати в базі даних колекцію, назву якої відповідає назві моделі у множині. Тому при роботі з даними моделі Proifle та User (додаванні, видаленні, редагуванні і отриманні об'єктів) mongoose буде звертатися до колекції «users» та «profile» . Якщо така колекція є в БД, то з нею буде йти взаємодія. Якщо такої колекції в базі даних немає, то вона буде створена автоматично. Другий параметр функції mongoose.model - власне схема.

Процес авторизації користувача

Авторизація– це процес керування рівнями та засобами доступу до веб-додатку. Для спрощеної розробки методу авторизації доцільно використовувати бібліотеки авторизації. У веб-додатку використовується Passport.js – це бібліотека для автентифікації Node.js. Надзвичайно гнучкий і модульний, Passport може бути непомітно вставлений в будь-який веб-додаток на основі Express. Повний набір стратегій підтримує автентифікацію з використанням імені користувача та пароля, Facebook, Twitter. Існує велика кількість способів та протоколів авторизації/автентифікації. У веб-додатку використовується спосіб авторизації/автентифікації за допомогою токенів.

Реалізація цього способу полягає в тому, що identity provider (IP) надає достовірні відомості про користувача в вигляді токена, а service provider

(SP) додаток використовує цей токен для ідентифікації, автентифікації і авторизації користувача.

На загальному рівні, весь процес виглядає наступним чином:

- Клієнт розпізнається в identity provider одним із способів, специфічним для нього (пароль, ключ доступу, сертифікат, Kerberos).
- Клієнт просить identity provider надати йому токен для конкретного SP-додатки. Identity provider генерує токен і відправляє його клієнту.
- Клієнт розпізнається в SP-додатку за допомогою цього токена.

Рисунок. 4.2. Приклад автентифікації «активного» клієнта за допомогою токена, переданого за допомогою Bearer схеми.

Існує кілька поширених форматів токенів для веб-додатків. У веб-додатку використовується JSON Web Token (JWT).

JSON Web Token (JWT) - містить три блоки, між якими ставлять крапку: заголовок, набір полів (claims) і підпис. Перші два блоки представлені в JSON-форматі і додатково закодовані в формат base64. Набір полів містить довільні пари ім'я / значення, до того ж стандарт JWT визначає кілька зарезервованих імен (iss, aud, exp і інші). Підпис може генеруватися за допомогою і симетричних алгоритмів шифрування, і асиметричних. Крім того, існує окремий стандарт, відписувався формат зашифрованого JWT-токена. Токени надають собою засіб авторизації для кожного запиту від

клієнта до сервера. Токени (і відповідно сигнатура токена) генеруються на сервері ґрунтуючись на секретному ключі (який зберігається на сервері) і в payload.

Passport.js підтримує можливість використання JWT для авторизації автентифікації у веб-додаток.

Конфігурація JWT Strategy у Passport.js:

```
const opts = {};
opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
opts.secretOrKey = keys.secretOrKey;
module.exports = passport => {
  passport.use(
    new JwtStrategy(opts, (jwt_payload, done) => {
      User.findById(jwt_payload.id)
        .then(user => {
          if (user) {
            return done(null, user);
          }
          return done(null, false);
        })
        .catch(err => console.log(err));
    })
  );
};
```

На основі конфігурації можна розробити процес отримання access token - використовується для авторизації запитів і зберігання додаткової інформації про користувача.

Процес отримання access token:

```
module.exports = function(req, res, next){
  // Get token from the header
  const token = req.header('x-auth-token');
  // Check if no token
  if (!token) {
      return res.status(401).json({msg: 'no token, auth denied'});
  }
  // Verity token
  try {
    const decoded = jwt.verify(token, config.get('jwtSecret'));
    req.user = decoded.user;
    next();
  } catch (err) {
    res.status(401).json({msg: 'Token is not valid'})
  }
}
```

Створення API

При використанні веб-додатку користувач повинен швидко отримувати інформацію з серверу та бази даних. Клієнтська частина відправляє запит на сервер і повертає відповідь на його запит, а браузер її приймає й відображає отриману інформацію. У веб-додатку використовується REST API. REST - скорочення від Representational State Transfer, що можна перекласти як «передача репрезентативного стану». Це стиль проектування розподілених систем за допомогою обмежень. Центральною абстракцією в REST є ресурс. Ресурс - це представлення віртуального об'єкту (такого як зображення), реального об'єкту (клієнт) чи колекції об'єктів. У веб-додатку розроблена REST API для обробки даних для авторизації, створення нових користувачів, отримання даних про певного користувача, редагування навантаження на викладачів, видалення користувачів та отримання даних про всіх користувачів. Лістинг представлений в додатку В.

4.2. Розробка клієнтської частини веб-додатку

Клієнтська частина відповідає за інтерфейс взаємодії між користувачем і веб-додатком. Для розробки інтерфейсу користувача використовується фреймворк React.js оснований на мові програмування Javascript. В основі розробки інтерфейсів є компонентна структура. Компонент це одна ізольована частина інтерфейсу з певним вмістом коду. Компоненти можуть взаємодіяти з собою і покликатись один на одного. На рисунку 4.3. зображана архітектура клієнтської частини веб-додатку.

Рисунок 4.3. Архітектура клієнтської частини веб-додатку

App.js – це корінь файл в якому прописана конфігурація веб-додатку, підключені всі залежності та створена маршрутизація веб-додатку. У React є своя система маршрутизації, яка дозволяє зіставляти запити до додатка з певними компонентами. Ключовою ланкою в роботі маршрутизації є модуль **react-router**, який містить основний функціонал по роботі з маршрутизацією. В системі маршрутизації кожен маршрут зіставляється з певним компонентом.

Router визначає набір маршрутів і, коли до додатка, приходить запит, то Router виконує зіставлення запиту з маршрутами. І якщо якийсь маршрут збігається з URL запиту, то цей маршрут вибирається для обробки запиту. Також для вибору маршруту визначено об'єкт Switch. Він дозволяє вибрати перший-ліпший маршрут і його використовувати для обробки. Маршрут відображення сторінки авторизації:

```
<Route exact path="/" name="Login Page" render={props => <Landing {...props}
/>} />
```

Кожен маршрут представляє об'єкт Route. Він має ряд атрибутів. Зокрема, тут для маршруту встановлюються чотири атрибути:

- **path**: шаблон адреси, з яким буде зіставлятися запитана адреса URL;
- **render** - той компонент, який відповідає за обробку запиту по цьому маршруту;
- **exact** - Цей кваліфікатор допускає тільки точний збіг маршруту з рядком запиту;
- **name** – назва шляху.

Для захисту веб-додатку потрібно захищати маршрути від несанкціонованого доступу. Для цього потрібно розробити захисний шлях.

Логічне відображення захисту шляху:

```
const PrivateRouter = ({ component: Component, auth, ...rest }) => (
  <Route
    {...rest}
    render={props =>
      auth.isAuthenticated === true ? (
        <Component {...props} />
      ) : (
          <Redirect to="/" />
        )
    }
  />
);
<PrivateRouter exact path="/workload" component={DefaultLayout} />
```

Login.js – це компонент авторизації користувача у веб-додаток. Він містить поля для вводу електронної адреси пароль та функціональну кнопку «Login». Логічна частина компонента відповідає за отримання даних з полів введення надіслання введених даних до серверу та отримання даних з серверу і перевіряє роль користувача. Лістинг авторизації представлений у додатку Г.

SideDrawer.js – компонент відповідає за графічне відображення елементів керування веб-додатком.

Alert.js – компонент відповідає за графічне відображення застережень та помилок.

TextFieldGroup.js – компонент відповідає за графічне відображення полів для введення даних.

MainTable.js – це компонент, який відповідає за основний функціонал веб-додатку. Компонент містить у собі компонент Profiles.js

Profiles.js – це компонент який відображає таблицю даних навантаження на викладачів кафедри, містить у собі гіпертекстову розмітку заголовків таблиці та логічне відображення інформації про навантаження на кожного викладача.

ProfileItem.js – цей компонент відповідає за відображення інформації про навантаження на кожного викладача.

MyProfile.js - це компонент який відображає головну інформацію про користувача. Містить у собі компонент ProfileAbout.js

ProfileAbout.js - це компонент який відображає загальну інформації про викладача.

SingleTeacher.js - це компонент який відображає, вибраного адміністратором, навантаження викладача.

Teachers.js - це компонент який відображає загальну інформацію про всіх користувачів.

4.3. Висновки до розділу 4

Отже, в цьому розділі детально була розглянута та побудована архітектура веб-додатку за допомогою певного стеку технологій а саме: серверної мови програмування Node.js, проаналізована архітектура роботи фреймворка React.js, каскадних таблиць стилів, мови програмування JavaScript та базою даних MongoDB. Також була реалізована та описана клієнтська та серверна частина веб-додатку і проаналізована схема баз даних.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

В результаті виконання даної випускної кваліфікаційної роботи було розроблено клієнт-серверну інформаційну системи розподілу навантаження на викладачів кафедри з використанням певного стеку технологій, а саме серверної мови програмування Node.js, JavaScript фреймворку React.js, каскадних таблиць стилів, мови програмування JavaScript та базою даних MongoDB.

Були визначені основні поняття і принципи обліку методичної, наукової роботи педагогічних і науково-педагогічних працівники закладів вищої освіти, та розглянуто формування загального обсягу навантаження. Розглянуто загальну характеристика об’єкту інформаційної системи та розглянуто сучасні програмні забезпечення розподілу навантаження на викладачів кафедр. Була детально проаналізована серверна та клієнтська частина інформаційної системи. Данна інформаційна система відповідає всім цілям вимогам згідно поставлених на нього задач. Поставлені завдання роботи успішно виконані і мета досягнута.

В ході розробки були:

- Отримано навички аналізу і проектування;
- вивчена теорія управління закладами вищої освіти;
- спроектована інформаційна система розподілу навантаження на викладачів кафедри;
- розроблена інформаційна система розподілу навантаження на викладачів кафедри на основі раніше отриманих проектних рішень;
- Впроваджені системи згідно поставлених завдань.

Впроваджені методи обробки даних та функції

запитів/відповідей між базою даних.

- Отримано навички проектування неструктурованих баз даних.

Отриманий досвід буде використаний для розробки веб-додатків спрямованих на розробку більш функціональних інформаційних систем.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman & Co. New York, NY, USA, 2017 – 445c.
2. Hultberg, T.H., Cardoso, D.M.: The teacher assignment problem: a special case of the fixed charge transportation problem. European Journal of Operational Research 2018, – 473c.
3. Emelechev, V.A., Perepelitsa, V.A.: The complexity of discrete multicriteria problems. Discrete Math. Appl., 2016 – 117c.
4. Shelley Powers Learning Node– Sebastopol CA : O'Reilly Media, 2016. – 288 c.
5. Alex MacCaw JavaScript Web Applications – Sebastopol CA : O'Reilly Media, 2011 – 257 c.
6. Addy Osmani Developing Backbone.js Applications - Sebastopol CA : O'Reilly Media, 2013 – 374c.
7. Kristina Chodorow MongoDB The Definite Guide – Sebastopol CA : O'Reilly Media, 2011 – 432c.
8. Zaozerskaya, L.A., Plankova, V.A.: The using discrete optimization models at development of an automatical knowledge checking system, 2018 – 47-52 c.
9. David Herron Node.JS Web Development - Third Edition – Birmingham : Packt, 2016 – 376c.
10. Mike Cantelon Node.js in Action / Marc Harter T.J/ Holowaychuk Nathan Rajlich - Shelter Island, New-York NY : Manning Publications, 2014 – 417c.

11. Ethan Brown Web development with Node and Express. Learning the JavaScript stack – Sebastopol CA : O'Reilly Media, 2014. – 329 c.

12. Valentin Bojinov RESTful Web API Design with Node.js - Second Edition – Birmingham : Packt, 2016 –146c.

13. Rick Darnell JavaScript Quick Reference / Mike Afergan - Indianapolis IN: Macmillan Publishing Co, 1996 – 208c.

14. Christopher Schmitt HTML5 Cookbook / Kyle Simpson – Sebastopol CA: O'Reilly Media, 2011 – 284c.

15. Peter Rob Database Systems: Design, Implementation, and Management, Eighth Edition / Carlos Coronel – New-York NY : Course Technology, 2009 – 235c.

16. Wesley Hales HTML5 and JavaScript Web Apps – Sebastopol CA : O'Reilly Media, 2012 – 172c.

17. Semmy Purewal Learning Web App Development – Sebastopol CA : O'Reilly Media, 2014 – 306c.

18. Simon Holmes Mongoose for Application Development – Birmingham : Packt, 2013 –142c.

19. Alex Banks Learning React: Functional Web Development with React and Redux -– Sebastopol CA : O'Reilly Media, 2017 – 220c.

20. Marc Garreau Redux in Action Will Faurot -– Sebastopol CA : O'Reilly Media, 2018 – 312c.

21. MongoDB – Режим доступу : https://www.mongodb.org. – Дата доступу : 29.04.2018.

22. JavaScript - Режим доступу : https://developer.mozilla.org Дата доступу : 30.04.2018.

23. Node.js - Режим доступу : :https://nodejs.org/ Дата доступу : 23.04.2018.

24. Introduction to Node & Express Режим доступу : https://medium.com/. Дата доступу : 10.03.2018

25. How to write powerful schemas in JavaScript - Режим доступу : https://medium.com/. Дата доступу : 02.04.2018