РЕФЕРАТ / ABSTRACT

Атестаційна робота магістра містить: 138 с., 61 рис.,4 табл., 25 джер.

HR-СИСТЕМА, NODE.JS, СУБД MONGODB, NODE EXPRESS, HTML, BOOTSTRAP, CSS ТА JAVASCRIPT.

Метою роботи є розробка інформаційної системи автоматизації HR-процесів з використанням сучасних технології та нереляційних систем управління даними із врахуванням показників узгодження реплік в базах даних NoSQL на етапі проектування інформаційних систем.

Дослідження проводилися на базі комплексного системного аналізу з використанням методів теорії масового обслуговування, теорії ймовірностей і математичної статистики, імітаційного моделювання, методології функціонального моделювання, інженерії знань, теорії множин, теорії алгоритмів та об'єктно-орієнтованого проектування.

В результаті роботи розглянуто методи управління реплікаціями в NoSQL системах, розроблено програмну реалізацію системи автоматизації HR-процесів, яка представляє собою веб-додаток з використанням Node.js та MongoDB, Node Express, який є веб-додатком для Node.js, а інтерфейс створений за допомогою HTML, Bootstrap, CSS та JS.

HR-SYSTEM, NODE.JS, MONGODB, NODE EXPRESS, HTML, BOOTSTRAP, CSS AND JAVASCRIPT.

The aim of the work is to develop an information system for automation of HR-processes using modern technologies and non-relational data management systems, taking into account the indicators of matching replicas in NoSQL databases at the stage of designing information systems.

The research was conducted on the basis of complex systems analysis using the methods of queuing theory, probability theory and mathematical statistics, simulation modeling, functional modeling methodology, knowledge engineering, set theory, algorithm theory and object-oriented design.

As a result, the methods of replication management in NoSQL systems are considered, the software implementation of the HR-process automation system is developed, which is a web application using Node.js and MongoDB, Node Express, which is a web application for Node.js, and the interface is created using HTML, Bootstrap, CSS and JS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ІС – інформаційна система

БД – база даних

СУБД – система управління базами даних

UML - Unified Modeling Language

NoSQL - not only SQL

HR - Human resources

ВСТУП

Актуальність теми. Сучасний стан автоматизації HR-процесів характеризується динамічним розвитком в сучасному світі, багато професій виходять на новий рівень.

Системи для автоматизації та діджиталізації HR-процесів набирають обороти, стають все популярнішими та пропонують все більш широкий функціонал.

Діджиталізація поступово перетворюється із стремління на оптимізацію процесів в структурі необхідності сучасного HR-а. Комплексні фахівці, що допомагають компаніям, активно розвиваються. Це дуже важлива тенденція у світі управління людськими ресурсами. Завдяки їй поступово приходить розуміння потреб розібрати HR-а та автоматизувати частину процесів, щоб звільнити ресурси для інших важливих завдань. Ведення таблиці та навчальних матеріалів можуть бути автоматизованими, у той час як для підтримки мотивацій та професійного росту співробітників важливо автоматизувати відносини з HR-спеціалістом.

Потреба в автоматизації управлінських HR-завдань виникла не так давно, тому, можна сказати, що даний напрямок ще знаходиться в розвитку. До вибору систем для автоматизації управлінської діяльності варто підходити з особливою обережністю також і тому, що управлінські функції специфічні в кожній організації, а значить необхідно орієнтуватися на максимальну відповідність існуючого в компанії алгоритму аналізу інформації та закладеного в системі.

Основна проблема, з якою стикаються фахівці з роботи з персоналом - це велика трудомісткість управління, величезна кількість завдань, функцій, процесів, якими необхідно оперативно і якісно керувати.

Спеціальні системи автоматизації управління персоналом формалізують і оптимізують всі процеси кадрової діяльності і дозволяють більш ефективно управляти людськими ресурсами. Автоматизована система управління HR-процесам дозволяє не просто систематизувати кадрове управління, а і здійснювати

моніторинг і контроль, в якому важко буде знайти місце для помилок, порушень і зловживань.

Тому, розробка інформаційної системи автоматизації HR-процесів є актуальним напрямком досліджень.

В процесі аналізу HR-систем встановлено важливість серверної частини, яка виступає засобом зберігання інформації. В якості серверної частини використовується СУБД. Виходячи із сучасних тенденцій, то особлива увага зараз приділяється NoSQL системам.

У таких системах дані зберігаються в вигляді таблиць, вони також припускають наявність схеми бази даних. Але при створені великих систем (Big Data) з використанням реляційних СУБД розробники стали зазнавати значних труднощів: при наявності великої кількості вузлів виникає проблема забезпечення необхідної відмовостійкості системи.

Як спроба вирішити накопичені проблеми реляційних баз даних з'явилися альтернативні засоби збереження та обробки даних, які отримали назва «бази даних NoSQL». Піонерами в цій галузі виступили дві компанії: Google і Amazon. В БД NoSQL для забезпечення високої відмовостійкості використовується багаторазова реплікація (копіювання) записів. Але бази даних NoSQL володіють недоліком: в цих системах не підтримується режим ведення транзакцій і блокувань, тому виникає проблема узгодження реплікацій.

Важливими показниками узгодження реплік в системах баз даних NoSQL є ймовірність читання застарілого запису за час поширення оновлень по вузлах системи, час очікування початку читання запису із оновлених серверів, число версій записів в базі даних NoSQL і час їх обробки, ймовірність відмови в доступі до запису БД та ін. Ці характеристики необхідно оцінювати на етапі проектування системи, тому що це дозволяє уникнути ручного підбору значень необхідних параметрів для великого числа типів записів БД на етапі налагодження системи і необхідності натурного моделювання екстремального навантаження на систему.

Так як технологія розробки інформаційних систем на основі баз даних NoSQL є досить новою, математичні моделі, необхідні для оцінки показників узгодження реплік, або відсутні, або є неадекватними.

Тому розробка адекватних математичних моделей і програмних засобів, що дозволяють на етапі проектування систем NoSQL оцінювати показники узгодження реплік і вибирати необхідні параметри, є актуальною задачею.

Зв'язок роботи з науковими програмами, планами, темами. Напрямок виконаних досліджень безпосередньо пов'язаний з науково-дослідним напрямком кафедри програмної інженерії Тернопільського національного технічного університету імені Івана Пулюя.

Мета і задачі дослідження. Метою роботи є розробка інформаційної системи автоматизації HR-процесів з використанням сучасних технології та нереляційних систем управління даними із врахуванням показників узгодження реплік в базах даних NoSQL на етапі проектування інформаційних систем.

Для вирішення поставленої мети вирішуються наступні завдання:

- аналіз предметної області управління HR-процесами на фірмах, виділення основних пріоритетних напрямків її розвитку;

- здійснити порівняльний аналіз відомих HR-систем, виділити їх основні переваги та недоліки;

- провести дослідження варіантів неузгодженості даних для кожного варіанта вирішуваних завдань, а також ймовірність читання неузгоджених даних за час поширення змін на етапі проектування системи;

- проаналізувати особливості узгодження реплік в базах даних NoSQL, а сосбливо MongoDB;

- дослідити відомі моделі і методів оцінки показників якості функціонування баз даних NoSQL;

- розробити моделей процесів узгодження реплік при оновленні будь-якого запису бази даних MongoDB;

- розробити математичну модель відмов і відновлення доступу до записів в базах даних MongoDB;

- здійснити проектування інформаційної системи автоматизації HR-процесів;

- програмно реалізувати інформаційну систему автоматизації HR-процесів з використанням Node.js та MongoDB;

- провести процедуру тестування системи та її практичну апробацію, включаючи атапи встановлення, розгортання, налаштування, розробка інструкції користувача;

- здійснити експериментальні дослідження моделей управління реплікаціями на етапі проектування системи автоматизації HR-процесів.

Об'єкт дослідження – процеси реалізації інформаційної системи управління HR-процесами.

Предмет дослідження – методи та програмні засоби реалізації інформаційної системи управління HR-процесами.

Методи дослідження. Дослідження проводилися на базі комплексного системного аналізу з використанням методів теорії масового обслуговування, теорії ймовірностей і математичної статистики, імітаційного моделювання, методології функціонального моделювання, інженерії знань, теорії множин, теорії алгоритмів та об'єктно-орієнтованого проектування.

Наукова новизна одержаних результатів. Розроблено модель процесів узгодження реплікацій, що дозволяють розрахувати ймовірність зчитування застарілого запису з бази даних NoSQL для режимів синхронного і асинхронного оновлення даних.

Практичне значення одержаних результатів. Для практичного використання отриманих теоретичних результатів розроблено інформаційну систему афтоматизації HR-процесів, яка базується на використанні сучасних інформаційних технологій.

Особистий внесок магістранта. Всі результати отримані автором самостійно.

Апробація результатів. Основні положення магістерського дослідження апробовані на Всеукраїнській школі-семінарі молодих вчених і студентів

«Комп’ютерній інформаційні технології», що проходила 30 листопада 2020 року у м. Тернополі на базі Західноукраїнського національного університету.

Публікації. Крашівський А. І. Розробка веб-системи з використанням NODE.JS та MONGODB на прикладі системи автоматизації HR-процесів [Текст] / А. І. Крашівський // Комп’ютерні інформаційні технології : матеріали школи-семінару молодих вчених і студентів СІТ’2020 [м. Тернопіль, 30 листопада 2020 р.] / відп. за вип. М. П. Дивак. - Тернопіль : ЗУНУ, 2020. - С. 23-24.

1 ОСОБЛИВОСТІ УПРАВЛІННЯ HR-ПРОЦЕСАМИ

1.1 Автоматизація HR-процесів

Сучасний стан автоматизації HR-процесів характеризується динамічним розвитком в сучасному світі, багато професій виходять на новий рівень. До прикладу, HR-менеджер давно перестав бути «кадровиком», і зараз представляє себе багатопрофільним спеціалістом. Сучасний HR-спеціаліст займається:

- розвитком бренду компанії;
- корпоративною культурою;
- оцінкаи ефективності співпраці;
- визначенням вимог до майбутніх співробітників, відібранням баз кандидатів;
- питаннями підвищення продуктивності співробітників;
- обліком відпусків та лікарняних;
- організацією митингів та опитувань;
- проведенням зустрічі 1:1 (один на один) та огляд результатів;
- підготовкою корпоративів та іншими аспектами.

Половина цих завдань передбачає щоденне заповнення таблиць та створення документів. Така робота дуже важлива для бізнесу, виділяється багато ресурсів для HR-спеціаліста. Маючи об’ємні завдання, HR висуває щоденне трактування їх виконаного часу, відображаючи інші, не менш важливі завдання.

Один з можливих варіантів покращення ситуацій - автоматизація рутинних завдань HR-менеджера. Системи для автоматизації та діджиталізації HR-процесів набирають обороти, стають все популярнішими та пропонують все більш широкий функціонал.

Діджиталізація поступово перетворюється із стремління на оптимізацію процесів в структурі необхідності сучасного HR-а. Комплексні фахівці, що допомагають компаніям, активно розвиваються. Це дуже важлива тенденція у світі управління людськими ресурсами. Завдяки їй поступово приходить розуміння

потреб розібрати HR-а та автоматизувати частину процесів, щоб звільнити ресурси для інших важливих завдань. Ведення таблиці та навчальних матеріалів можуть бути автоматизованими, у той час як для підтримки мотивацій та професійного росту співробітників важливо автоматизувати відносини з HR-спеціалістом.

Сучасні системи автоматизації HR-процесів включають реалізацію наступних характеристик:

- працювати з управління відвідуваністю (облік відпусток, відгулів, лікарняних та ін.);
- моніторити настрій співробітників;
- відслідковувати адаптаційні процеси, розвиток співробітників, мотивацію та систему бонусів;
- інтегруватися з мессенджерами та порталами за пошуком роботи;
- складатиь детальні аналітичні звіти;
- створити якісні умови для рекрутингу;
- економія коштів компанії за рахунок автоматизації рутинних завдань та багато іншого.

Наприклад, система сама буде вести відгули, хвороби, відпустки та відсутність за іншими причинами. Співробітник із свого особистого кабінету заповнює заяву відпустку, і він або визначається автоматично, або HR отримує повідомлення та може скорегувати дату вручну. Не потрібно тратити час на окрему зустріч, додаткові документи та позбутися бюрократії, все робиться за пару кліків.

Не секрет, що настрій впливає на якість роботи. Моніторинг настрою важливо, а як це робити кожен день? Представте собі компанію, де кілька сотень людей. HR просто фізично не може подорожувати до кожного і за чашкою кави невзначай дізнаватися, як робота, так і все в порядку. Система моніторингу дозволяє відслідковувати, досліджувати співробітників і як його настрій впливає на працездатність. Працівник заходить у систему та привітний екран пропонує йому вибрати настрій та прокоментувати його. HR отримує графік по всіх співробітниках. Тратується всу пару хвилин, а результат переходить всі очікування. Співробітник може описати все, що йому подобається і не виправляти на робочому

місці, а HR тримає руку на пульсі компанії, не намагаючись оббігати всі окремі відділи в надії зібрати інформацію. Можливо сказати, що такий спосіб менш душевний, чим особисте спілкування, але у великій компанії спілкуватися з усіма особисто немає можливості.

Відслідковування адаптаційних процесів, мотивацій та систем бонусів реалізується за допомогою можливості визначити вітальні, умовні, адаптаційні, вихідні співбесіди, зустрічі 1:1, створити шаблонні анкети для оцінки продуктивності роботи співробітників і так далі.

Інтеграція з мессенджерами та сервісами з пошуку роботи дозволяє заповнити та розмістити нові вакансії в декілька кліків. Як правило, автоматизовані системи дозволяють розміщувати вакансії на особистому сайті компанії та різних зовнішніх сервісах.

Зручний інструмент для планування та моніторингу роботи завжди доступний в HR-системі. Система дозволяє візуалізувати робочі процеси, відмітити виконані завдання, додати нові, переглянути завдання, які знаходяться на стадіях виконаних, і так далі. Чим більше у вас завдань, тим більш складний розвиток в завданнях та їх категоріях. Автоматизація цього процесу дозволяє легко та швидко сортувати та аналізувати існуючі завдання. Все це сприяє суттєвій економії часі HR-а, яка тепер може покращити увагу людей, не шукаючи ресурсів на рутинні завдання.

1.2 Огляд і аналіз існуючих аналогів, що реалізують функції предметної області

Існують різні системи для автоматизації HR-процесів. Здійснимо аналіз найвідоміших рішень, відзначимо основні переваги та недоліки. Система BambooHR, головне вікно якої представлено на рисунку 1.1.

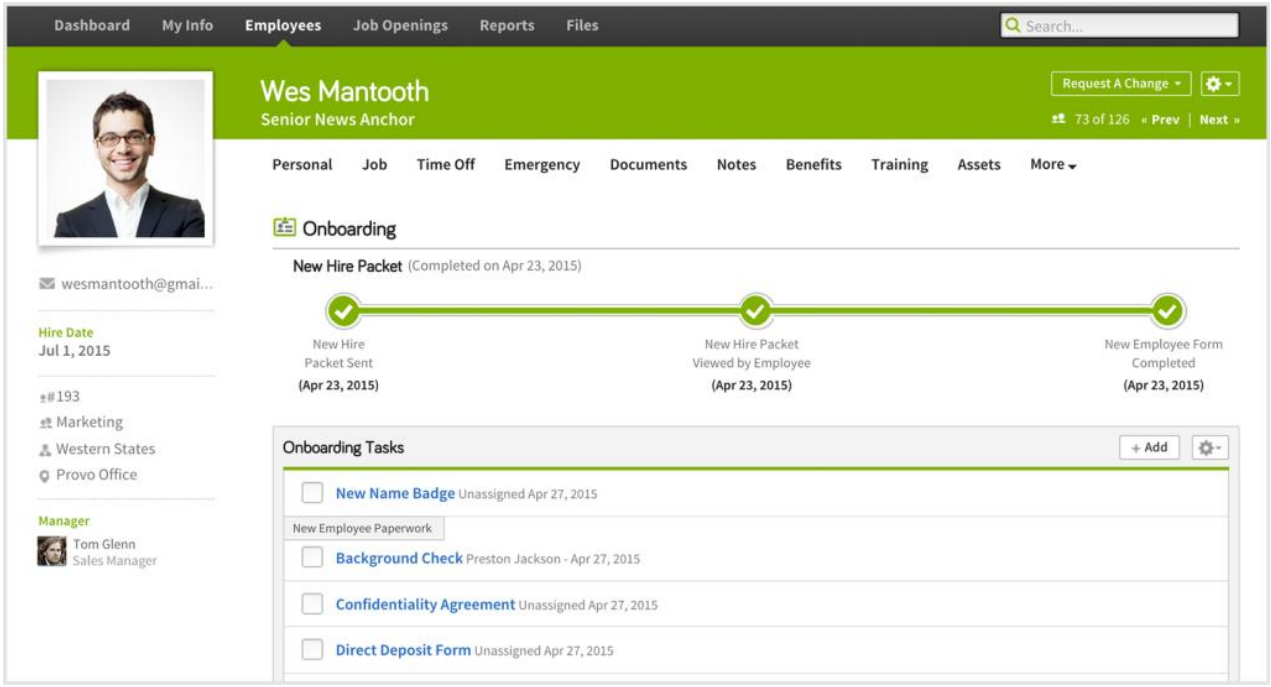


Рисунок 1.1 - Система BambooHR

Система для HR-фахівців, що пропонує комплексні рішення для управління персоналом. Найкраще підходить для зростаючих, невеликих і середніх компаній. Можливості BambooHR:

- система пошуку і відстеження кандидатів;
- система відстеження винагород;
- інтеграція з іншими інструментами (Zapier, Okta, OneLogin, Indeed, Greenhouse) і детальна аналітика;
- вбудований календар подій;
- моніторинг і управління навчанням співробітників;
- детальна документація (FAQ);
- доступ через мобільні платформи.

Серед недоліків системи, то тут варто відзначити відсутність автоматизації наступних бізнес-процесів:

- призначеного для користувача рейтингу серед співробітників;
- записів цілей для команди;
- історії оцінок співробітника.

Hurma System (рисунок 1.2). Нещодавно з’явилося на ринку рішення, яке поєднує в собі процеси рекрутингу, HR і OKR (Objectives and Key Results). В системі є пакети для компаній різних обсягів, підходить, як для зростаючих команд, так і для великих. Можливості Hurma System:

- моніторинг настрою співробітників;
- організація структури компанії у вигляді дерева;
- синхронізація з календарем Google, повідомлення про події компанії;
- особистий кабінет для кожного співробітника;
- парсинг резюме;

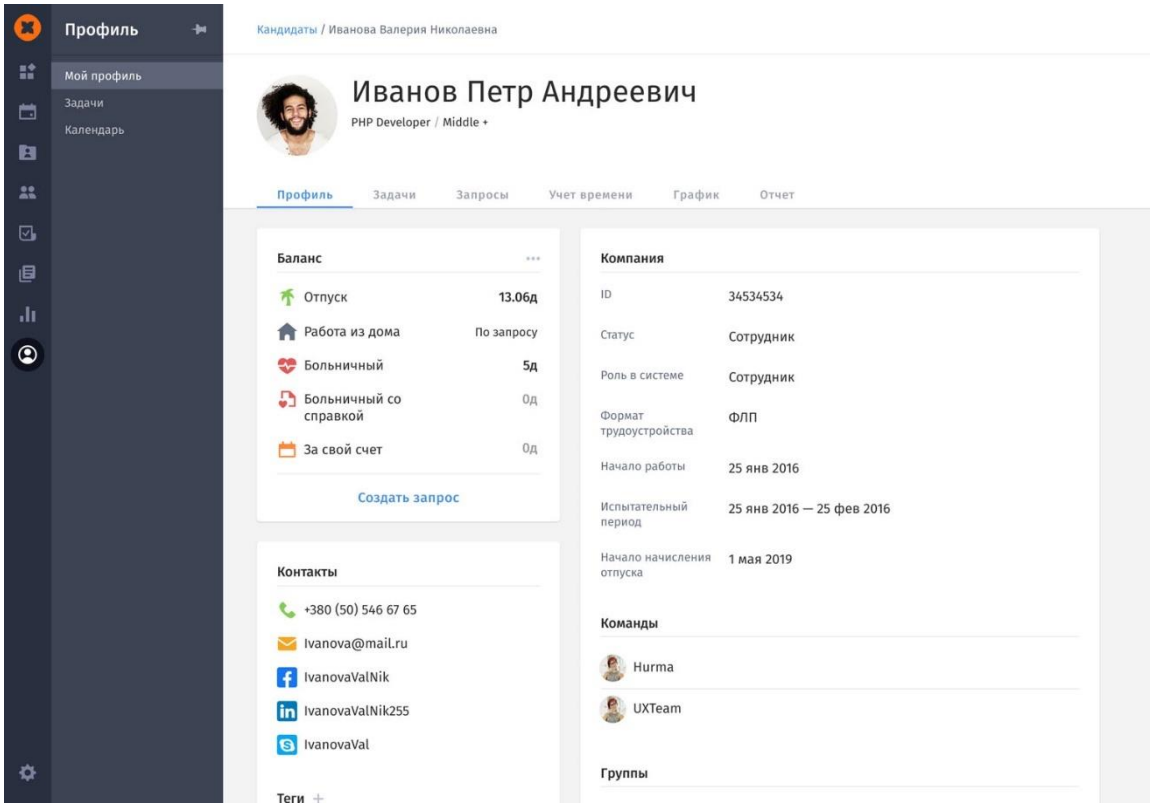


Рисунок 1.2 - Система Hurma System

- автоматизація Performance review, Welcome / Adaptation / Probation / Exit interview;
- можливість запросити one-to-one з HR та керівництвом;
- робота з OKR;
- absence management;

- база вакансій компанії і база кандидатів для рекрутера;
- публікація вакансій на сайті компанії;
- інтеграція з LinkedIn;
- ведення статистики рекрутингу та HR-процесів.

Недоліки системи:

- в фільтрі кандидатів не можна виділити за конкретними напрямами;
- сортування вакансій тільки за назвою і датою створення;
- не можна коментувати завдання за погодженням відпусток, віддаленої роботи та лікарняних днях.

Система Workable (рисунок 1.3). Система управління рекрутинг-процесами, що поєднує в собі моніторинг кандидатів (Applicant Tracking System) і рекрутинг-платформу з сильним двигуном пошуку, заснованим на машинному навчанні. Можливості Workable:

- швидкий парсинг резюме;

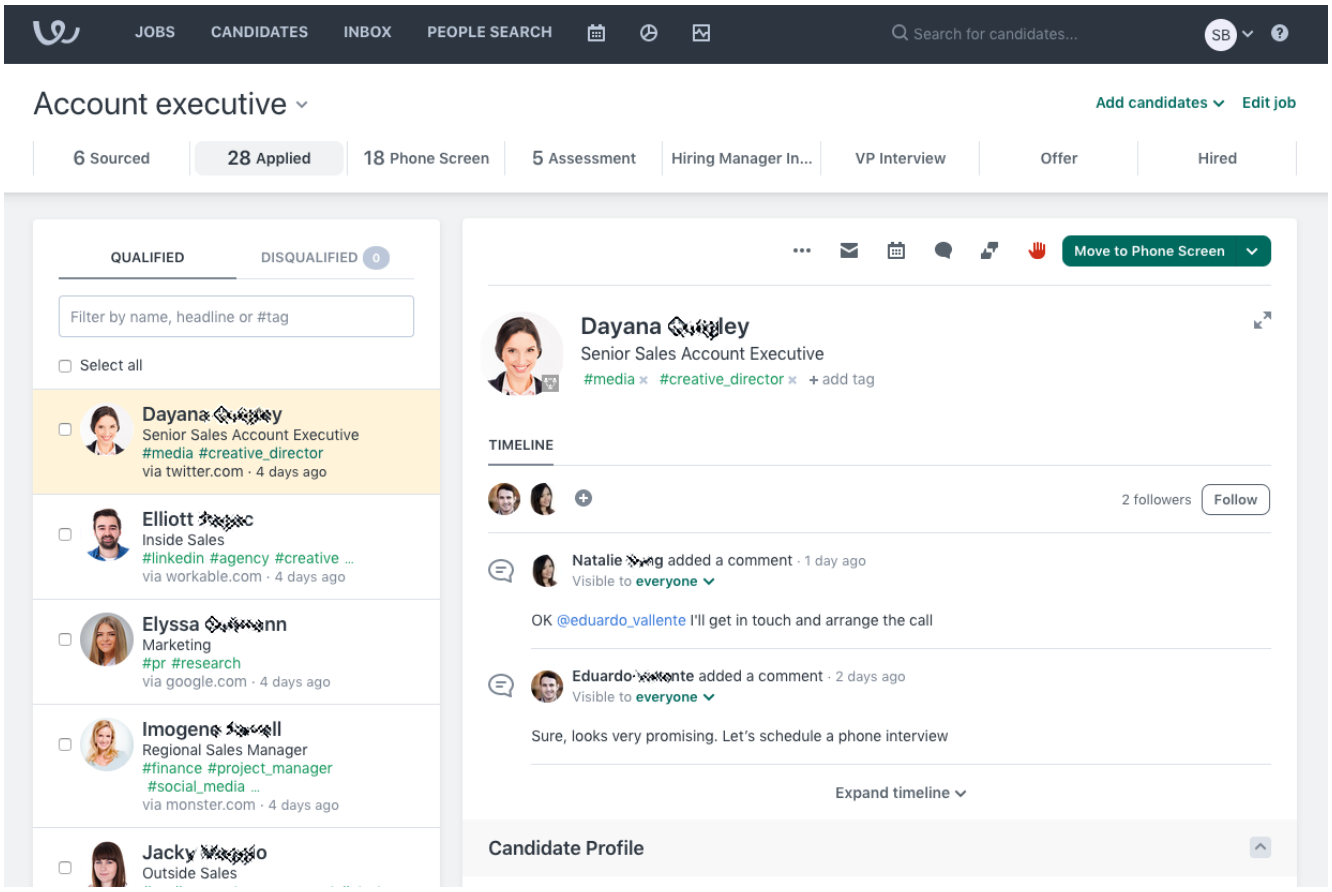


Рисунок 1.3 - Система Workable

- автоматизація пошуку співробітників (конструктор для створення анкет, резюме та вакансій);
- інтеграція з job-порталами;
- інтеграція з LinkedIn і можливість пошуку кандидатів через соціальні мережі;
- брендування інтерфейсу системи під фірмовий стиль вашої компанії;
- синхронізація календаря і розклад завдань для HR-відділу;
- імпорт існуючих баз даних;
- персональний менеджер, служба підтримки через телефон і email;
- мобільний додаток;
- розширення під Google Chrome для швидкого пошуку кандидатів.

Система Zoho People (рисунок 1.4). Програмне забезпечення для управління персоналом, призначене для малих і середній підприємств. Володіє простим інтерфейсом і працює відразу «з коробки». Можливості Zoho People:

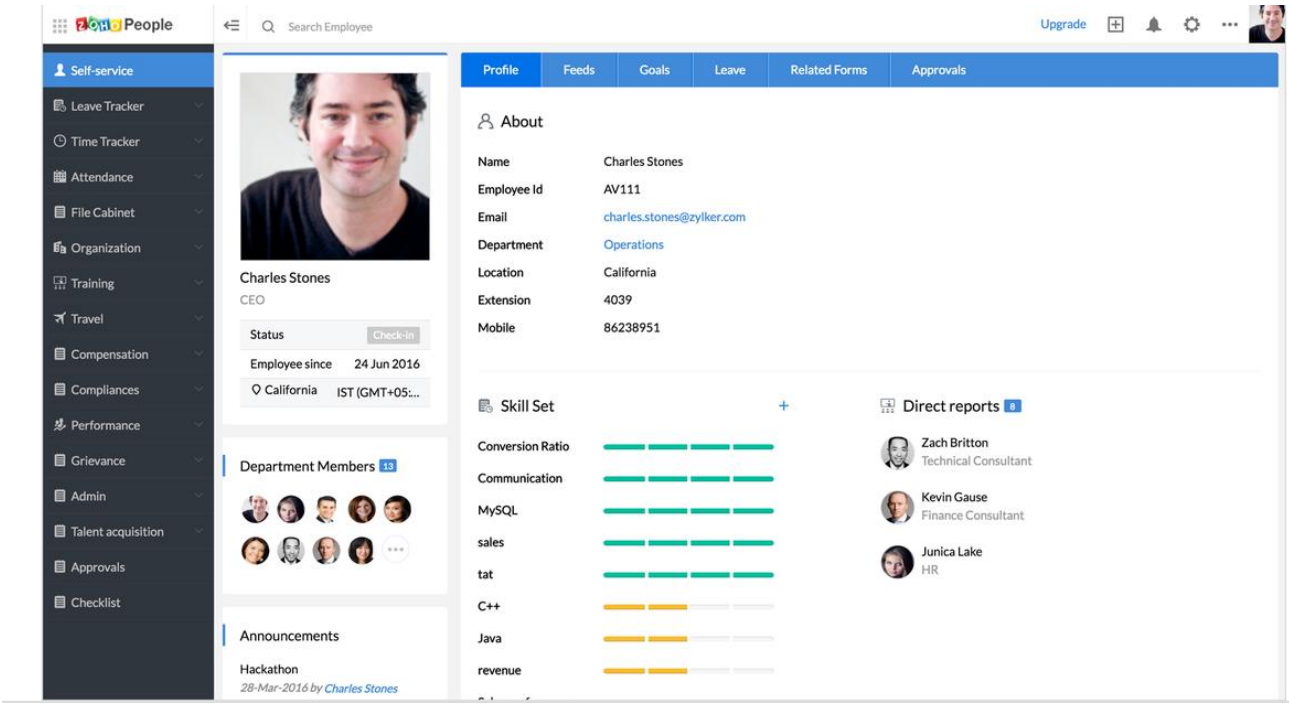


Рисунок 1.4 - Система Zoho People

- наявність функціоналу для ведення absence management;
- шаблони для стандартних HR-документів;

- наявність мобільного додатку та web-версії;
- трекер відпрацьованого часу;
- ведення аналітики для HR;
- функції для проведення Performance review;
- електронний підпис;
- внутрішня аналітика і звітність.

Робота HR-фахівця вкрай важлива для компанії. HR-менеджер виступає буфером між співробітниками і керівниками, розвиває бренд компанії, покращує корпоративну культуру, займається мотивацією і адаптацією співробітників. Крім цього, потрібно вести документацію, відстежувати всі лікарняні, відпустки, відрядження, проводити співбесіди, призначати події і так далі. Всі ці завдання необхідно виконувати на високому рівні і без помилок, так як часто від HR-а залежить робота інших фахівців.

Автоматизація дозволяє зменшити обсяг рутинних справ, поставивши їх на потік, і змістити фокус на завдання більш високого рівня, такі як підвищення лояльності, мотивації та продуктивності команди, адаптація нових співробітників, розвиток корпоративної культури та багато іншого.

1.3 Аналіз узгодження реплік в базах даних NoSQL

В процесі аналізу HR-систем встановлено важливість серверної частини, яка виступає засобом зберігання інформації. В якості серверної частини використовується СУБД. Виходячи із сучасних тенденцій, то особлива увага зараз приділяється NoSQL системам.

В останні кілька років в області обробки даних домінували реляційні СУБД. У таких системах дані зберігаються в вигляді таблиць, вони також припускають наявність схеми бази даних. Але при створені великих систем (Big Data) з використанням реляційних СУБД розробники стали зазнавати значних труднощів:

1) ускладнилася процедура агрегування даних, тому що це вимагає читання записів з великого числа пов'язаних таблиць (виникла проблема втрати відповідності);

2) виникло протиріччя між необхідністю зберігання великих обсягів неструктурованих даних і необхідністю їх якось структурувати за допомогою розробки схеми бази даних;

3) для зберігання великих обсягів інформації необхідно купувати дорогі спеціалізовані аппаратно-програмні комплекси паралельних систем баз даних (Teradata, Sun Oracle Database Machine і ін.);

4) при наявності великої кількості вузлів виникає проблема забезпечення необхідної відмовостійкості системи.

Як спроба вирішити накопичені проблеми реляційних баз даних з'явилися альтернативні засоби збереження та обробки даних, які отримали назва «бази даних NoSQL». Піонерами в цій галузі виступили дві компанії: Google і Amazon. В БД NoSQL для забезпечення високої відмовостійкості використовується багаторазова реплікація (копіювання) записів. Але бази даних NoSQL володіють недоліком: в цих системах не підтримується режим ведення транзакцій і блокувань, тому виникає проблема узгодження реплікацій.

Важливими показниками узгодження реплік в системах баз даних NoSQL є ймовірність читання застарілого запису за час поширення оновлень по вузлах системи, час очікування початку читання запису із оновлених серверів, число версій записів в базі даних NoSQL і час їх обробки, ймовірність відмови в доступі до запису БД та ін. Ці характеристики необхідно оцінювати на етапі проектування системи, тому що це дозволяє уникнути ручного підбору значень необхідних параметрів для великого числа типів записів БД на етапі налагодження системи і необхідності натурного моделювання екстремального навантаження на систему.

Так як технологія розробки інформаційних систем на основі баз даних NoSQL є досить новою, математичні моделі, необхідні для оцінки показників узгодження реплік, або відсутні, або є неадекватними.

Незважаючи на різноманітність баз даних NoSQL, в процесі функціонування вони виконують деякі загальні функції, пов’язані з узгодженням реплік:

- розміщення реплік записів БД в кластері і забезпечення їх узгодження при оновленні запису;

- узгодження версій репліки (зведення кількох версій записів до одного запису в процесі ведення версій записів);

- узгодження (відновлення) реплік після усунення збою в вузлі.

У базах даних NoSQL в основному використовуються два способи розміщення і оновлення реплік: за принципом «головний-підлеглий» (master-slave) і «по кільцю »(ring).

Перший спосіб має на увазі зберігання даних на master-вузлі і їх реплікацію на slave-вузли. Всі зміни виконуються на master-вузлі, і вони зберігаються в пам’яті цього вузла. Slave-вузли періодично опитують master вузел, читають накопичені зміни і зберігають їх у своїй пам’яті.

Прикладами таких баз даних є MongoBD, HBase, Neo4j та ін. При розміщенні даних «по кільцю» необхідно визначити, скільки в ньому буде секцій (v-вузлів). Ці секції розподіляються по серверам (по кільцю). На рисунку 1.5 показаний приклад [14]. Тут визначені 64 секції і вони по колу розміщені по трьом фізичним серверам A, B, C. База даних виділить кожному серверу 21 або 22 секції (64/3).

При включенні запису в базу даних обчислюється хеш ключа, за допомогою якого визначається номер секції (v-вузла), за яким визначається сервер, де буде зберігатися цей запис. Інші репліки записи (їх число дорівнює N-1) зберігаються на наступних (N-1) серверах, розташованих по кільцю за годинниковою стрілкою щодо першого сервера. Прикладами баз даних з розподілом по кільцю є Riak, Amazon Dynamo і ін. Використання віртуальних вузлів (v-вузлів) має наступні переваги [26]:

- якщо вузол стає недоступним (через збої або плановому обслуговувані), навантаження рівномірно розподіляється між вузлами;

- коли вузол знову стає доступним або в систему додається новий вузол, цей вузол приймає на себе приблизно рівне навантаження від кожного з інших доступних вузлів;

- кількість віртуальних вузлів, за які відповідає реальний вузол, може визначатися виходячи з потужності сервера, що дозволяє вирівняти навантаження в неоднорідній інфраструктурі.

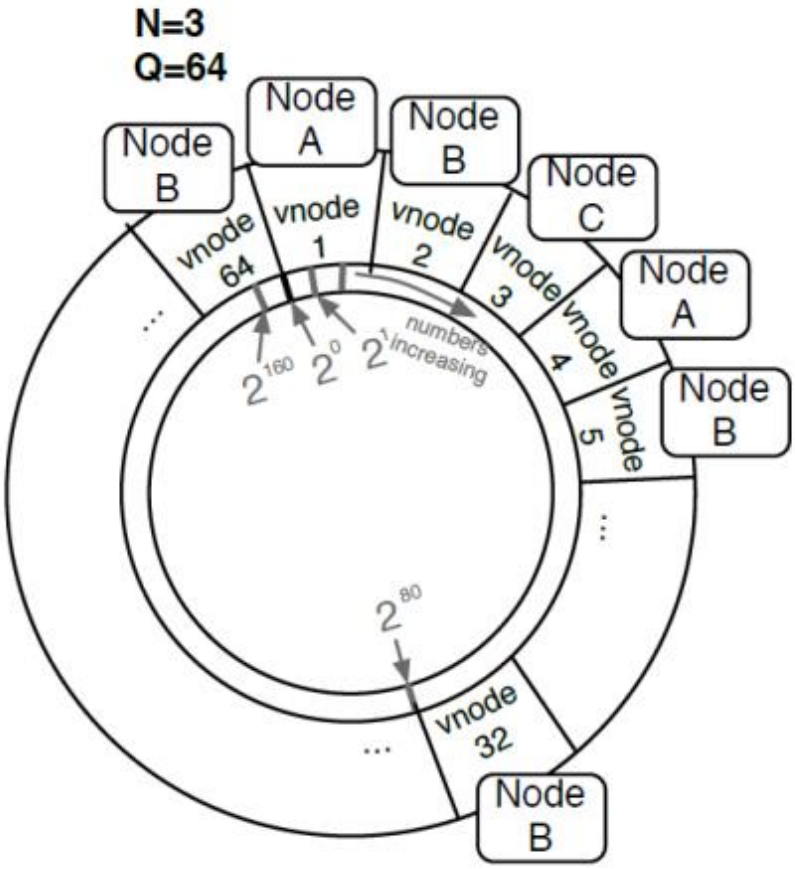


Рисунок 1.5 - Кільце з 64 v-вузлами і трьома серверами

Реплікація використовується для підвищення продуктивності і відмовостійкості. При оновленні запису інші репліки оновлюються не відразу, а протягом вікна неузгодженості - часу, який проходить від моменту поновлення запису на будь-якому вузлі до моменту завершення оновлень копій запису на всіх інших вузлах. Розрізняють такі види узгодженості [17]:

- сувора узгодженість - будь-який запит на читання завжди поверне останнім оновлення запису;
- слабка узгодженість - система не гарантує, що запит на читання завжди поверне останнє оновлення запису;
- узгодженість в кінцевому рахунку - особлива форма слабкої узгодженості: система зберігання гарантує, що, якщо згодом не буде ніяких нових оновлень записів, в кінцевому рахунку запит на читання поверне останнє оновлення.

Узгодженість тісно пов'язана з доступністю системи, а також з стійкістю до втрати зв'язності. Цей зв'язок виклав Ерік Брювер в теоремі CAP [18]. У теоремі стверджується, що можна створити розподілену систему, яка буде 1) узгодженою (Consistent), 2) доступною (Available) і 3) стійкою до втрати зв'язності (Partition tolerant), але одночасно можна гарантувати тільки два з цих трьох властивостей. При узгодженості в кінцевому рахунку гарантуються властивості 2 і 3.

Надалі в роботі розглядається сувора узгодженість реплік і узгодженість реплік в кінцевому рахунку (КР-узгодженість). Введемо такі позначення:

N - кількість вузлів, на які в кінцевому рахунку буде репліковано запис (з деякою затримкою);

W - кількість вузлів (реплік), на які дані повинні бути фактично записані перед тим, як користувачу (або додатком) буде відправлена відповідь про успішне завершення операції (якщо $W < N$, то система все ще продовжує реплікувати дані на решту N-W вузлів);

R - кількість вузлів, від яких база даних очікує відповіді для успішного завершення читання записів.

Відсутність блокувань дозволяє одночасно читати і змінювати один і той же запис бази даних на різних вузлах. Це призводить до конфліктів, які розрішуються або штатними засобами NoSQL, або вручну.

Першим, найпростішим способом вирішення конфліктів є використання тимчасової мітки, якою забезпечується кожен запис. При виникненні конфліктів

перевага віддається найновішому запису. Але як відзначається в [18], такий підхід складно реалізувати в кластері вузлів.

Другим способом вирішення конфліктів є ведення версій записів, реалізована за допомогою вектора годин (Vector Clock - VC) [14, 26]. Вектор годин - це послідовність пар <користувач, номер версії запису для цього користувача>, яка описує порядок поновлення цього запису.

На рисунку 1.6 показаний приклад ведення вектора годин (в дужках показаний вектор годин запису). Запис *D* (наприклад, якийсь документ) оновлюється користувачами A1, A2, A3. Перші два поновлення виконуються користувачем A1 послідовно. Далі користувачі A2, A3 одночасно читають і оновлюють цей запис (випадковий збіг). У базі даних зберігаються дві версії запису: D1 і D2. При читанні документа D користувачеві A1 повертаються дві версії запису з одним і тим же ключем (D1 і D2). Він вносить зміни, наприклад, поєднує в собі оновлення, виконані користувачами A2, A3. У базі зберігається одна узгоджена версія запису з вектором годин, що включає ідентифікатори трьох користувачів.

Переваги вектора годин: відсутність єдиної точки відмови системи, тому що при використанні тимчасових міток в записах необхідно виконувати точну синхронізацію часу з одним еталоном.

Недоліки вектора годин: відсутність можливості автоматично вирішувати конфлікти, а також збільшення довжини вектора годин при багаторазовому оновленні запису. Однак в NoSQL існують механізми усічення вектора годин. Наприклад, в системі Riak можна задавати частоту обрізання вектора на рівні сегмента, а також максимальний розмір (довжину) вектора годин [19].

Наступний підхід усунення конфліктів полягає в створенні нечасової мітки (це може бути хешування вмісту, GUID, лічильник і т.д.), яка повертається користувачеві разом з необхідними даними з СУБД. Після виконання змін користувачем система порівнює значення мітки, отримане від користувача, з тим, яке зберігається в базі даних.

Якщо значення не збігаються, операція відхиляється. Користувач повинен знову прочитати і оновити запис. Даний підхід використовується в системі CouchDB [30].

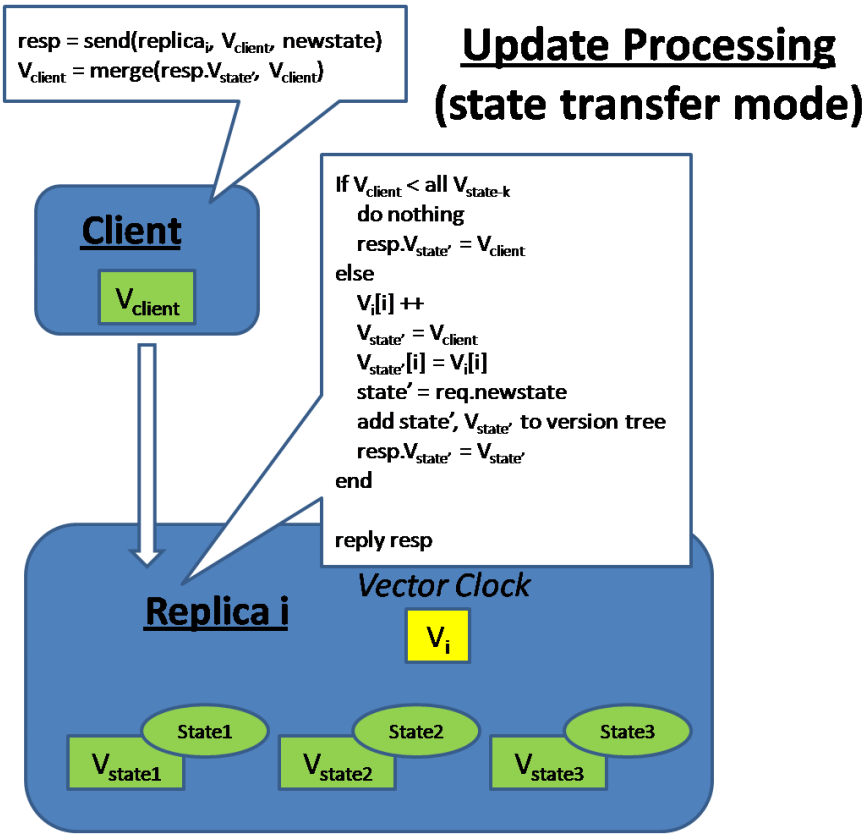


Рисунок 1.6 - Приклад ведення вектора годин

Наведемо приклади неузгодженості даних, що призводять до появи декількох версій записів, і способи їх усунення:

- кілька співробітників одночасно коректують екземпляр одного і того ж документа (запису) - наприклад, пропозиція щодо розвитку підприємства; дозвіл неузгодженості – об'єднання (вибір) коригувань керівником групи.

- кілька експертів одночасно оцінюють екземпляр одного і того ж документа - наприклад, blind-оцінка статті, поданої на конференцію; дозвіл неузгодженості - вибір оцінки головою комісії.

- кілька співробітників одночасно працюють з одним екземпляром документа, причому один співробітник може бачити результати роботи іншого

співробітника - наприклад, кілька авторів працюють над однією статтею; може з’явитися кілька версій записів, неузгодженість усуває керівник групи авторів.

- користувачі обговорюють якусь одну подію в блозі. При багатофазній обробці запиту за технологією MapReduce вихідні дані однієї фази є вхідними даними другої фази - наприклад, при виконанні операції Join при доступі до структурованих даних або при виконанні складних запитів в пошукових системах (репліки вхідних даних наступної фази можуть бути неузгоджені); цю неузгодженість усуває NoSQL (узгодженість, заснована на потенційному причинно-наслідкового зв’язку).

Додаток генерує послідовність пов'язаних записів, вони зберігаються в репліках не одночасно і в іншій часовій послідовності - наприклад, результати індексації документів в пошукових системах; неузгодженість реплік усуває NoSQL (узгодженість, заснована на явному причинно-наслідкового зв’язку).

З наведених вище прикладів видно, що забезпечення узгодженості дуже важливо для стабільного та належного функціонування системи. Одночасна робота призводить до конфліктів, які можуть бути дозволені, наприклад, керівником групи на основі вектора годин. Знання оцінки вікна неузгодженості сприяло б в цій ситуації запобіганню конфліктів.

У разі багатофазної обробки неузгодженості бути не повинно, тому що вихідні дані однієї фази використовуються як вхідні дані іншої. Знання затримки реакції системи при забезпеченні суворої узгодженості дозволяє виконати точну оцінку часу виконання всього багатофазного завдання.

При одночасній роботі великої кількості користувачів з одним записом бази даних число версій цього запису може бути велике. Тому виникає задача оцінки навантаження на користувача в залежності від числа користувачів, одночасно працюючих із записом: оцінка числа версій записів, часу їх обробки користувачем і ін. За результатами цих оцінок можна дати рекомендації про максимально можливому числі користувачів, одночасно які беруть участь в обговоренні.

1.4 Аналіз відомих моделей і методів оцінки показників якості функціонування баз даних NoSQL

Іноді на практиці набір параметрів *N, W, R* буває недостатнім для управління узгодженістю даних. Це відбувається при виникненні необхідності підвищити якість узгодженості при заданому рівні доступності та наоборот. В [20] розглядається проблема поділу властивостей безпеки і довговічності в розподілених сховищах даних з використанням «bolt-on» шару. Даний шар передбачає причинну узгодженість. Причинна узгодженість - це узгодженість, заснована на причинно-наслідкового зв'язку <сталося - до>. Таким чином, всі операції запису описують причинну історію. Даний підхід гарантує строгу узгодженість.

Причинні зв'язки найчастіше описуються в двох видах: потенційний і явний причинний зв'язок. В потенційному зв'язку все записи, які можуть вплинути на інший запис, повинні бути видні до того, як стане видно цей останній запис. У статті [10] представлена архітектура «bolt-on» шару наступним чином.

На кожну клієнтську машину пропонується встановити прошарок (shim), що містить метадані і локальне сховище. Клієнт звертається до даного прошарку, який в свою чергу звертається до сховища даних, узгодженим в кінцевому рахунку. Даний шар обмежує порушення узгодженості, які може побачити клієнт. Сховище даних відповідальне за розподіл процесів записів: для того, щоб прочитати нові версії інших процесів, кожен shim відправляє відповідний запит в сховище даних.

Локальна пам'ять містить узгоджений набір записів, з якими можуть проводитися операції читання і додавання в будь-який час без порушення обмежень безпеки. Даний набір записів позначається як «причинна вирізка даних» $x1 \rightarrow y1 \rightarrow z1$ (causal sat). Нехай інший клієнт зберіг запис $y2 \rightarrow NULL$. При пошуку останньої версії запису $y2$ прошарок shim повинен переконатися, що в сховищі був запис $y1$. Інакше причинний зв'язок $x1 \rightarrow y2$ буде загублено. У статті описані алгоритми роботи з shim, а також процедури перевірки цілісності

причинних зв'язків. Пропонована архітектура схематично зображена на рисунку 1.7.

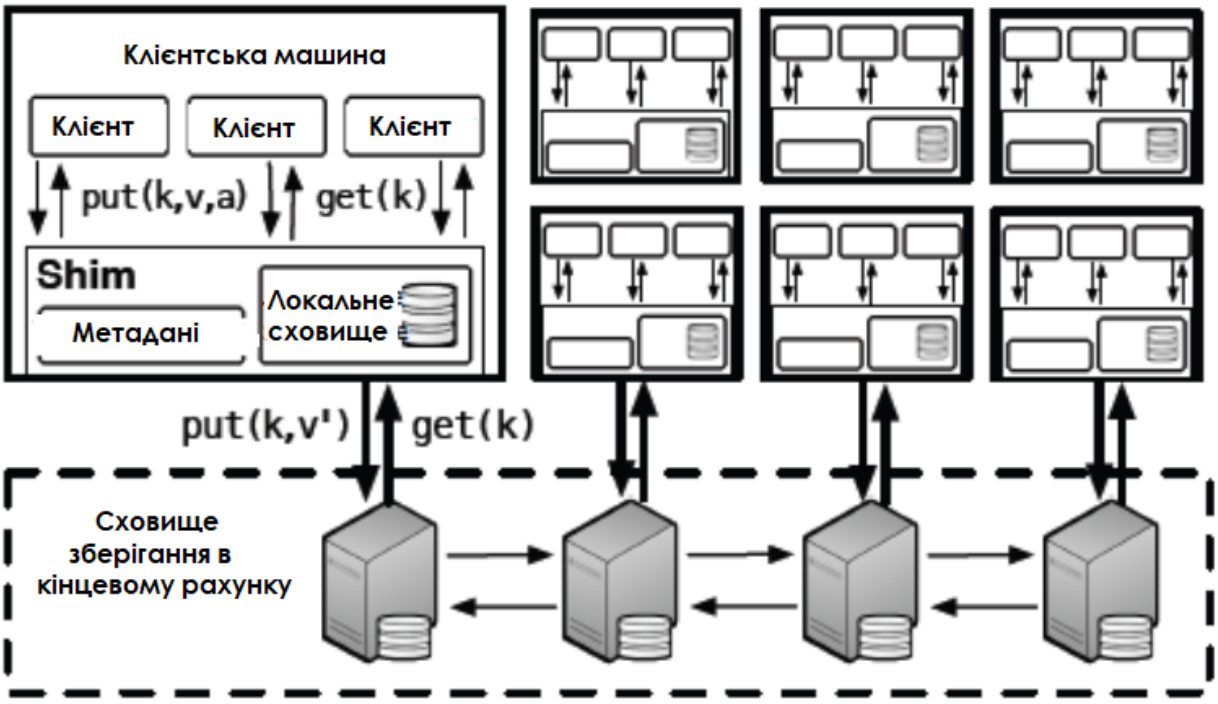


Рисунок 1.7 - Схема bolt-on сховища

Перевагою розглянутого підходу є підвищення рівня узгодженості до суворої узгодженості завдяки введенню додаткового прошарку на клієнтській машині. Недоліком є перенесення частини функцій бази даних на клієнтську машину, а, отже, зростання обсягу збережених даних, наявність додаткової затримки на синхронізацію локального і загального сховища даних.

Такий підхід можна використовувати для вузького кола додатків, наприклад, для уточнення цілей при пошуку даних. Узгодженість в кінцевому рахунку не розглядалася.

У статті [14] пропонується метод LibRe узгодженості даних в базах даних NoSQL. Метод ґрунтується на узгодженості в кінцевому рахунку. Завдання полягає в тому, щоб поєднувати високу доступність даних з високим рівнем узгодженості даних. Наприклад, якщо $N = 3, W = 1, R = 1$, то має місце КР-узгодженість,

тому що $W + R \leq N$. Підхід, пропонований в статті, дозволяє посилити узгодженість. LibRe розшифровується як Library For Replication.

Підхід полягає в наступному. Керуючий модуль LibRe розташовується поруч з балансувальником завантаження. Балансувальник завантаження перед кожною операцією запису/поновлення передає LibRe набір вузлів, на яких можливо виконати операцію. Після виконання операції LibRe зберігає деякий час інформацію про те, який вузол її виконав. Для кожної операції читання LibRe повертає набір вузлів, які зберігають актуальні для поточного запиту набори даних. На рисунку 1.8 представлено розташування LibRe в системі.

Перевагою цього підходу є підвищення рівня узгодженості до суворої узгодженості завдяки введенню додаткового реєстру, який деякий час зберігає інформацію про те, на якій репліці відбувалося оновлення того чи іншого запису. Недоліком є наявність деякої затримки, необхідної для визначення потрібної репліки. Реєстр може стати «вузьким місцем».

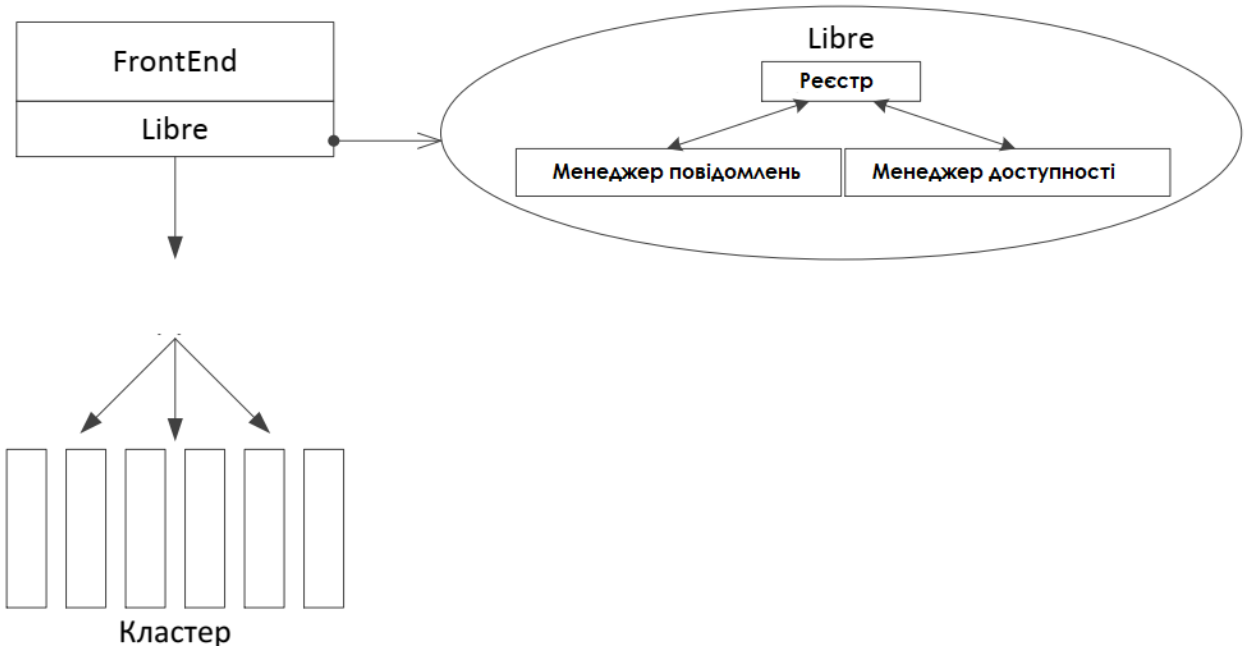


Рисунок 1.8 - Розташування libre в системі

В роботі [12] описується механізм PAXOS забезпечення узгодженості в розподілених базах даних. Підхід грунтується на узгодженій (Consensus)

реплікації. Алгоритм узгодження реплік в класичному варіанті (Paxos basic) полягає в наступному:

1. Конкретна репліка (сервер) призначається в якості координатора.

2. Координатор вибирає запис і розсилає її всім реплікам для підтвердження, що приймає сервер може прийняти запис або відмовити.

3. Як тільки більшість реплік прийняли новий запис, узгодження досягнуто і всім реплікам висилається команда про підтвердження операції (commit).

При необхідності пункти 1-3 можуть бути повторені (через відмову мережі). Координатор може також відмовити. Але paxos не вимагає, щоб координатор був єдиним, в будь-який момент координатором може стати інша репліка.

1.5. Постановка задачі дослідження

Інформаційна система автоматизації HR-процесів повинна бути простою і зрозумілою у використанні, а саме повинен бути зручною пошук інформації, зрозумілий текст, зручний інтерфейс.

Програмне забезпечення повинно враховувати процеси управління персоналом на підприємстві, де воно впроваджується: ведення структури організації, ведення списку кандидатів, ведення кадрового плану по підрозділах і посад, ведення етапів інтерв'ю з кандидатами, парсер даних з сайтів кадрових служб (work.ua), надавати звітність по виконанню кадрового плану і стану кандидатів на роботу. Повинна існувати можливість швидкого переходу по пунктах меню і швидкого доступу до необхідної інформації. Система повинна бути відмовостійкою та надійною.

Виходячи з описаного вище аналізу, були формалізовані наступні задачі магістерського дослідження:

- аналіз предметної області управління HR-процесами на фірмах, виділення основних пріоритетних напрямків її розвитку;

- здійснити порівняльний аналіз відомих HR-систем, виділити їх основні переваги та недоліки;

- провести дослідження варіантів неузгодженості даних для кожного варіанта вирішуваних завдань, а також ймовірність читання неузгоджених даних за час поширення змін на етапі проектування системи;

- проаналізувати особливості узгодження реплік в базах даних NoSQL, а сосбливо MongoDB;

- дослідити відомі моделі і методів оцінки показників якості функціонування баз даних NoSQL;

- розробити моделей процесів узгодження реплік при оновленні будь-якого запису бази даних MongoDB;

- розробити математичну модель відмов і відновлення доступу до записів в базах даних MongoDB;

- здійснити проектування інформаційної системи автоматизації HR-процесів;

- програмно реалізувати інформаційну систему автоматизації HR-процесів з використанням Node.js та MongoDB;

- провести процедуру тестування системи та її практичну апробацію, включаючи атапи встановлення, розгортання, налаштування, розробка інструкції користувача;

- здійснити експериментальні дослідження моделей управління реплікаціями на етапі проектування системи автоматизації HR-процесів.

2 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ АВТОМАТИЗАЦІЇ HR-ПРОЦЕСІВ

2.1 Розробка моделей процесів узгодження реплік при оновленні будь-якого запису бази даних

У 1-му розділі зазначалося, що в базах даних NoSQL використовуються два способи узгодження реплік:

- узгодженість в кінцевому рахунку ($W + R \leq N$);

- сувора узгодженість ($W + R > N$).

Перший спосіб забезпечує швидкий доступ до даних, але при цьому можливо читання неактуальних записів. Цей метод використовується для забезпечення доступу до записів БД наступних типів: рекламні відомості, записи в блогах, новинні стрічки та ін.

Другий спосіб забезпечує гарантоване читання останнього поновлення запису, але при цьому можлива затримка читання. Цей метод застосовується для організації доступу до записів наступних типів: залишок і вартість товару в інтернет-магазині, чат, кеш користувача, сесійні дані та ін.

На показники якості узгодження реплік впливають такі параметри:

N - число реплік запису.

W - число реплік, для яких запис повинен бути збережений до повернення відповіді про успішне завершення операції запису.

R - число реплік, з яких запис повинен бути зчитано до повернення відповіді про успішне завершення операції читання.

В цьому розділі вирішуються наступні завдання:

1. Для способу узгодження реплік в кінцевому рахунку розробляється модель, що дозволяє оцінити ймовірність того, що вимога на читання отримає R неоновлення записів БД за час поновлення $N - W$ реплік записів, тобто що клієнт отримає застарілий запис (синхронний і асинхронний режими поширення оновлень).

2. Для способу суворого узгодження реплік розробляється модель, що дозволяє розрахувати характеристики випадкового часу очікування вимогою на читання закінчення поновлення W реплік і часу читання запису з урахуванням цього очікування.

Визначимо синхронний і асинхронний режими поширення оновлень записів БД по її репліках. Синхронний режим означає, що після отримання запиту на оновлення даних координатор послідовно направляє оновлення наступних репліці після завершення поновлення попередньої (рисунок 2.1а).

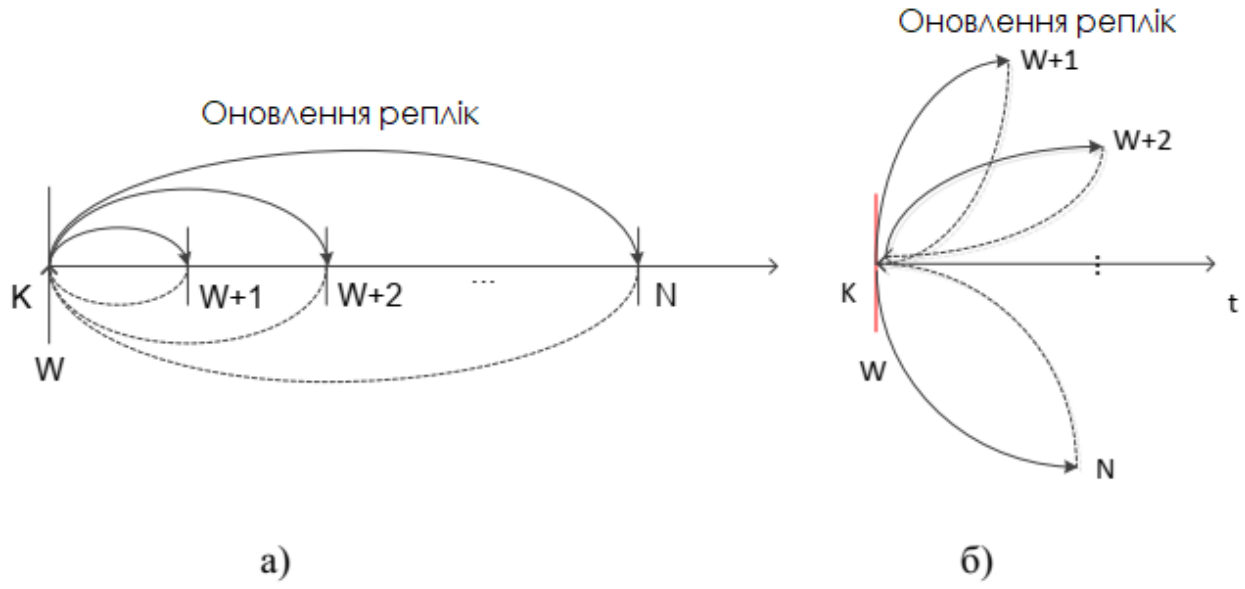


Рисунок 2.1 - Варіанти поширення змін: а) синхронний режим; б) асинхронний режим

При асинхронному режимі координатор направляє оновлення наступної репліки, не чекаючи завершення поновлення попередньої (рисунок 2.1б). Час поновлення реплік при асинхронному режимі менше, ніж при синхронному, тому що поширення змін виконується паралельно. На рисунку 2.1 оновлювані репліки позначаються $1,2,\ldots,N$, буквою К позначений координатор.

1. Синхронний режим. На рисунку 2.2 представлена модель неузгодженості даних для випадку узгодженості в кінцевому рахунку ($W + R \leq N$) [8, 9]. Вхідний потік вимог на читання з однієї репліки приймається пуассонівським з параметром

$\lambda, \Psi i\ (s)$ - перетворення Лапласа-Стілтьєса функції розподілу ймовірностей часу поновлення i-ої репліки.

Рисунок 2.2 - Модель узгодження реплік в кінцевому рахунку ($W + R \leq N$, синхронний режим)

У кожен момент часу з сумарною інтенсивністю $(W + i)\lambda$ надходять вимоги на читання з уже оновлених реплік і з інтенсивністю $(N - W - i)\ \lambda$ - з неоновлення реплік. Завдання полягає в тому, щоб оцінити ймовірність P того, що за випадковий інтервал поновлення реплік надійде хоча б одна вимога і по ньому буде прочитано застарілий запис в R репліках.

Для вирішення поставленого завдання спочатку пропонується отримати похідну функцію $Q_{W+i+1}(z)$ числа вимог на читання запису, які прийшли за випадковий час оновлення $(W + i + 1)$ -ї репліки цього запису і по ньому були прочитані R записів з її $N - W - i$ неоновлених реплік $(i = 0 \ldots N - W - 1)$.

$\psi_{W+i+1}(s)$ - перетворення Лапласа-Стілтьєса функції розподілу ймовірностей часу поновлення $(W + i + 1)$-ї репліки. Позначимо через g ймовірність того, що надійшла вимога на читання з неоновленої репліки отримає $R - 1$ неоновленних записів з інших реплік.

$$Q_{W+i+1}(z) = \psi_{W+i+1}(s)\big(\lambda \cdot g_i(N - W - i)(1 - z)\big), \tag{2.1}$$

$$g_i = \begin{cases} \frac{C_{N-W-i-1}^{R-1}}{C_{N-1}^{R-1}}, \text{якщо } N - W - i \geq R \\ 0, \text{ інакше} \end{cases} \tag{2.2}$$

Формула (2.2) доводиться, виходячи з класичного визначення ймовірності - це відношення сприятливого числа комбінацій до загальної кількості комбінацій. В даному випадку сприятливе число комбінацій - це число варіантів, при яких після надходження вимоги на читання записів з неоновленою репліки буде зчитано з БД $(R-1)$ неоновлення записів з решти $(N-W-i-1)$ неоновлених реплік. Загальне число комбінацій - це число варіантів, при яких можна вважати дані з $(R-1)$ реплік цього запису з залишившихся $(N-1)$ реплік (оновлених і неоновлених реплік).

З (2.2) отримаємо: $Q_{W+i+1}(0)$ - це ймовірність, що за час поновлення $(W+i+1)$ -ої репліки не надійдуть вимоги, за якими будуть прочитані R записів з $N-W-i$ неоновлених реплік, $1-Q_{W+i+1}(0)$ - це ймовірність, що за час оновлення $(W+i+1)$ - ої репліки надійде хоча б одна вимога на читання і по ньому буде прочитано R записів з $(N-W-i)$ неоновлених реплік.

Таким чином, ймовірність, що за час поновлення $N-W$ реплік надійде хоча б одна вимога на читання пари <ключ/значення> і по ньому буде прочитано R записів з неоновлених реплік, дорівнює:

$$P=(1-Q_{W+i}(0)+\sum_{i=2}^{N-W}\left(1-Q_{W+i}(0)\right)\prod_{j=1}^{i-1}Q_{W+i}(0) \qquad (2.3)$$

Вираз (2.3) випливає з формули повної ймовірності. Це і є ймовірність, що клієнт прочитає застарілий запис за час поширення оновлень записів по її $N-W$ реплікам.

Синхронний режим поновлення реплік підтримує база даних NoSQL Riak. Цей режим підтримує і система Hadoop. Тільки тут поновлення поширюються послідовно від попередньої репліки до наступної.

На рисунку 2.3 представлена модель узгодження реплік для випадку $W+R\leq N$ (асинхронний режим) [18, 19]. Вхідний потік вимог на читання з однієї репліки також приймається пуассонівським з параметром λ.

На рисунку 2.3 введені наступні позначення: a_i - випадкова мережева складова часу поновлення i-ої репліки, b_i- випадкова локальна складова часу поновлення i-ої репліки. У кожен момент часу з інтенсивністю λ незалежно один від одного надходять вимоги на читання записів БД з кожної репліки. Випадкові проміжки часу між початком поновлення t_0 і моментом надходження вимоги на читання з і-ої репліки позначимо через ξ_i. Завдання полягає в тому, щоб оцінити ймовірність H, що за час поновлення $N - W$ реплік надійде хоча б одна вимога на читання і по нього буде прочитано запис з її неоновлених реплік ($R = 1$).

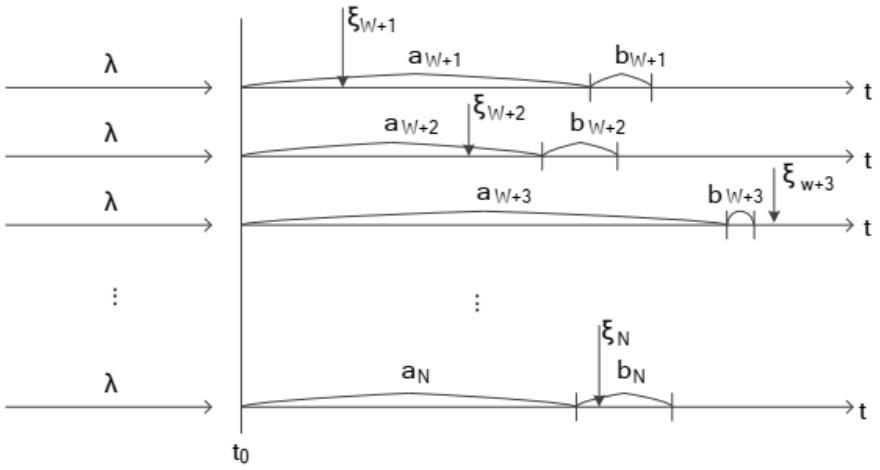


Рисунок 2.3 - Модель узгодження реплік в кінцевому рахунку ($W + R \leq N$, асинхронний режим)

Асинхронний режим оновлення реплік підтримує база даних NoSQL Amazon Dynamo, а також бази даних з типом реплікації «master-slave» і «Master-master».

2.2. Аналіз моделей узгодження реплік в кінцевому рахунку

Нижче наведені результати аналізу ймовірності, що клієнт прочитає застарілий запис за час розповсюдження оновлень запису по її $N - W$ реплік, для синхронного режиму поширення змін (формула (2.3)).

Характеристики ресурсів (інтенсивності обробки) були отримані за допомогою програми синтетичних тестів AIDA64 [15]. Розрахунки були виконані при наступних значеннях параметрів ресурсів.

1. Процесор - Mobile DualCore Intel Core i5-2450M, 2900 MHz. Для обраного процесора виміряне значення числа процесорних циклів, виконуваних в секунду, так само $\mu_P = 2900 \cdot 10^6$ (1/с).

2. Зовнішня пам'ять - Momentus 5400 640423 Seagate <ST9640423AS> 5400rpm 16Mb; інтенсивність введення/виведення даних на диск дорівнює μ_{D1P} =130·1024·1024 (байт/с).

3. Оперативна пам'ять - DDR3-1333 PC3-10667. Інтенсивність читання даних з ОП дорівнює μ_{ns} =9842·1024·1024 (байт / с).

4. Продуктивність локальної мережі всередині сегмента дорівнює 1 Гбіт/с; інтенсивність передачі даних по шині локальної мережі дорівнює μ_n = = 125·106 (Байт/с).

5. Продуктивність мережі між її сегментами становить 128 Мбіт/с; інтенсивність передачі даних по шині мережі, що з'єднує підмережі, дорівнює μ_{ns} = 16·106 (байт/с)

На рисунку 2.4 показані залежності ймовірності, що клієнт прочитає застарілий запис за час розповсюдження оновлень записів по її $N - W$ реплік, від інтенсивності вхідних запитів (вимог) при різних значеннях *N* (синхронний режим). Загальна кількість фізичних вузлів - 20, вузли розділені на два сегменти мережі по 10 вузлів в кожному. Число віртуальних вузлів (*V*-вузлів) - 64. Довжина «*k*» поля ключа запису становить 20 байтів, довжина «*v*» поля значення дорівнює 512 байтів. $W = R = 1$. У базах даних NoSQL, наприклад, RIAK [18] число реплік *N* кожного запису БД за замовчуванням дорівнює 3. У цьому випадку навіть при великих λ ймовірність дорівнює 0.03 (надійність узгодження реплік становить майже дві дев'ятки: 0,97).

При великих *N* ймовірність може досягати 0.3 при великих значеннях інтенсивності надходження вимог на читання. Розглянемо вірогідність, що клієнт прочитає застарілий запис за час розповсюдження оновлень записів по її N-W реплік, для асинхронного режиму поширення змін (формула (2.3)).

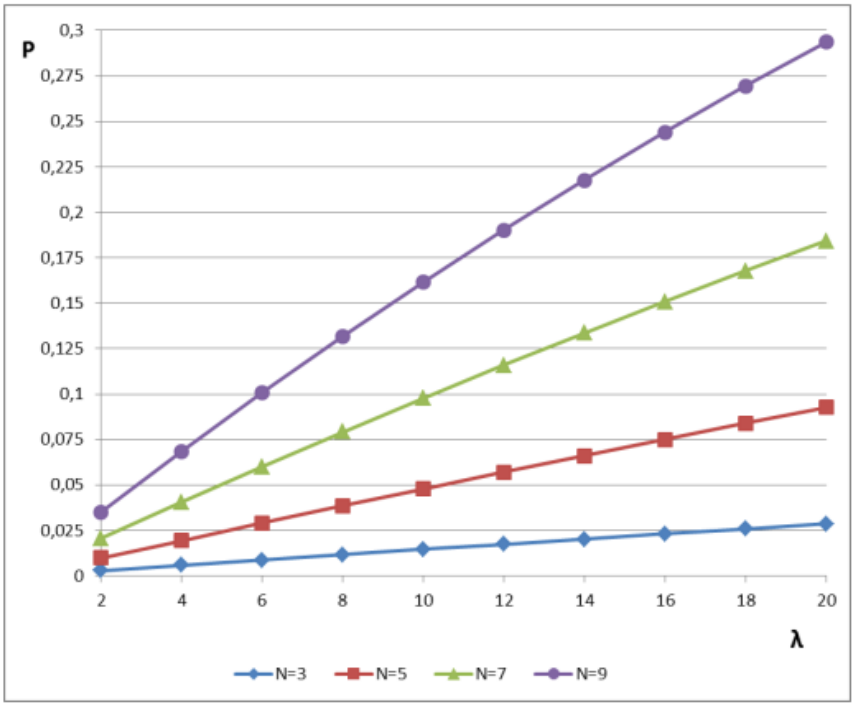


Рисунок 2.4 - Залежності ймовірності, що клієнт прочитає застарілий запис за час розповсюдження оновлень запису по її *N-1* реплік, від інтенсивності запитів на читання λ

Характеристики ресурсів збігаються з описаними раніше. На рисунку 2.5 показані залежності ймовірності, що клієнт прочитає застарілий запис за час розповсюдження оновлень записів по її *N-W* реплік, від інтенсивності вхідних запитів (вимог) при різних значеннях *N* (Асинхронний режим оновлення реплік).

У випадку $N = 3$ навіть при великих λ ймовірність не перевищує 0.0004 (надійність узгодження становить майже чотири дев'ятки: 0,9996). Порівняємо ймовірності доступу до неузгоджених даних при синхронному і асинхронному режимах поширення змін.

У таблиці 2.1 наведені значення ймовірності, що клієнт прочитає застарілий запис за час розповсюдження оновлень запису по $N - W$ реплік, для різних величин інтенсивності вхідних запитів при великих значеннях N (7 і 9) для синхронного і асинхронного режимів. Тут $W = R = 1$.

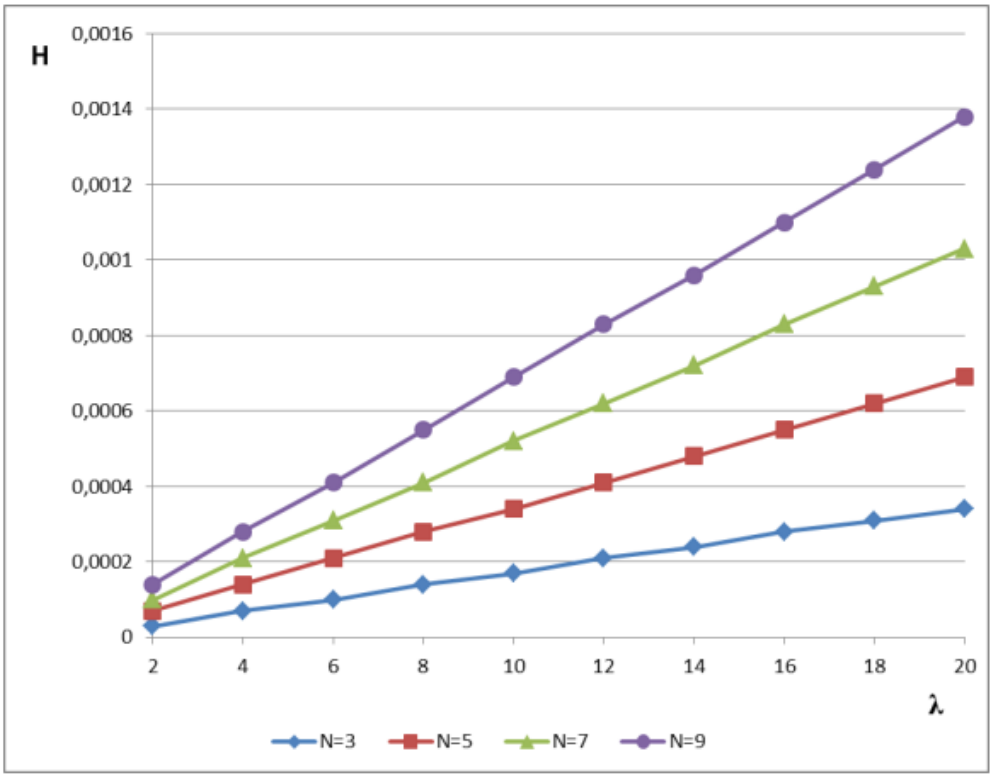


Рисунок 2.5 - Залежності ймовірності, що клієнт прочитає застарілий запис за час розповсюдження оновлень записів по її *N*-*W* реплік, від інтенсивності запитів на читання λ

З таблиці видно, що різниця в можливостях, розрахованих з урахуванням синхронного і асинхронного режимів досягає кілька порядків. Нижче наведені результати модельних експериментів по оцінці часу очікування вимогою на читання закінчення поновлення W реплік (формула (2.3)). Розрахунки були виконані для наступних значень характеристик ресурсів:

1. Продуктивність локальної мережі всередині сегмента дорівнює 100 Мбіт / с; інтенсивність передачі даних по шині локальної мережі дорівнює μ_n = = 12.5·106 (байт/с).

2. Продуктивність μ_{ns} мережі між її сегментами не враховувалася (відсутні підмережі). Інші характеристики ресурсів збігаються з відповідними значеннями. На рисунку 2.6 показані залежності математичного сподівання (МС) часу очікування вимогою на читання відновлення $W = N/2 + 1$ реплік від інтенсивності вхідних запитів (вимог) при різних значеннях N.

Таблиця 2.1 - Експериментальні розрахунки

	Синхронний режим		Асинхронний режим	
	N=7	N=9	N=7	N=9
λ	P	P	P	P
2	0.03970	0.06705	0.0001	0.00014
4	0.07753	0.12896	0.00021	0.00028
6	0.11357	0.18618	0.00031	0.00041
8	0.14794	0.23910	0.00041	0.00055
10	0.18071	0.28810	0.00052	0.00069
12	0.21199	0.33349	0.00062	0.00083
14	0.24184	0.37558	0.00072	0.00096
16	0.27034	0.41464	0.00083	0.00110
18	0.29757	0.45091	0.00093	0.00124
20	0.32358	0.48462	0.00103	0.00138

Довжина «*k*» поля ключа запису становить 20 байтів, довжина «*v*» поля значення дорівнює 2048 байтів.

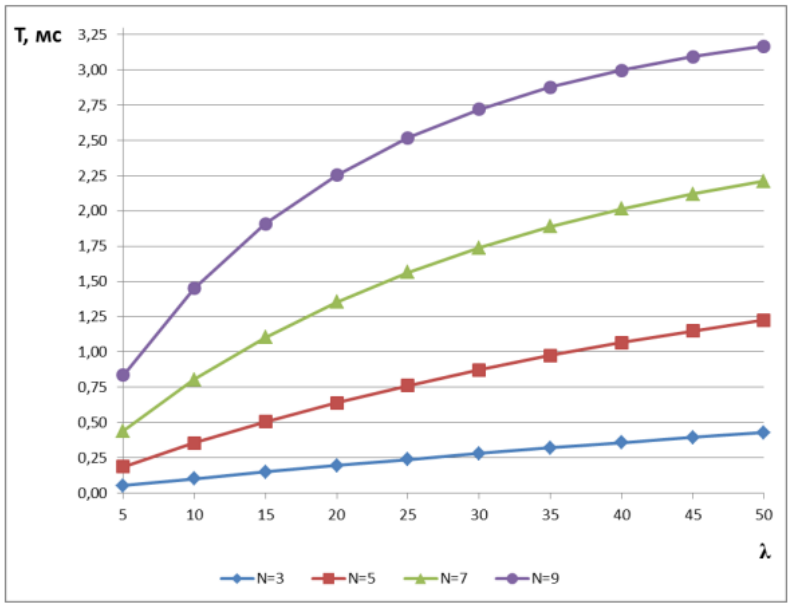


Рисунок 2.6 - Залежності математичного сподівання часу очікування вимогою на читання закінчення відновлення W реплік від інтенсивності запитів.

З рисунку 2.6 видно, що час очікування вимоги на закінчення відновлення W реплік при значенні $N = 3$ вимірюється в десятих частках мілісекунди. Однак при великих N і великій інтенсивності на читання цей час може досягати 3.2 мс.

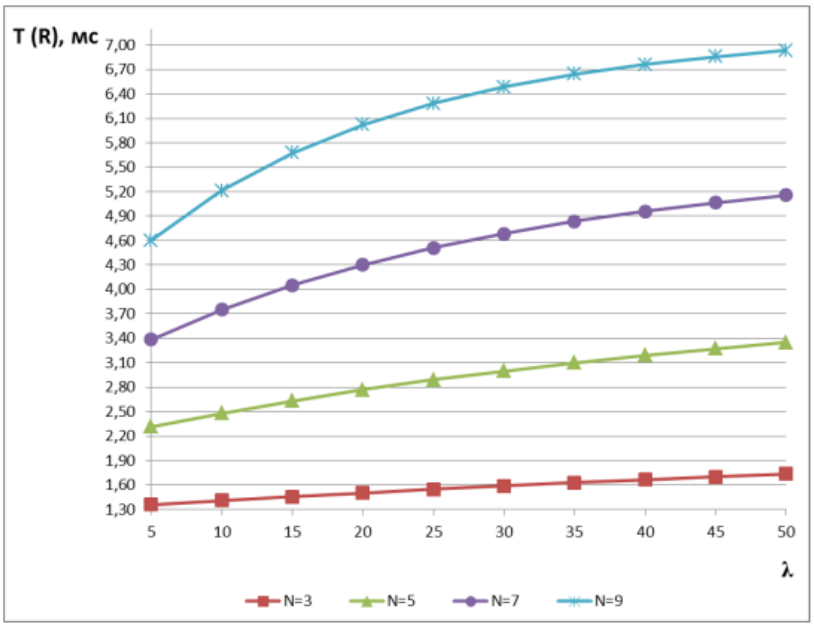


Рисунок 2.7 - Залежності МС часу читання R реплік з урахуванням часу очікування вимог на читання закінчення поновлення W реплік від інтенсивності запитів λ *(1 / с)* .

На рисунку 2.7 показані залежності МС часу читання R реплік з урахуванням часу очікування вимог на читання закінчення поновлення W реплік ($W = R = N/2 + 1$) від інтенсивності вхідних запитів при різних значеннях N .

2.3 Математична модель відмов і відновлення доступу до записів в базах даних NoSQL

Розглянемо кластер, що складається з U вузлів. Нехай кожен вузол виходить з ладу з інтенсивністю δ ($1/\delta$ – середній час напрацювання на відмову одного вузла). Вузол обслуговується ремонтною бригадою з інтенсивністю μ ($1/\mu$ - середній час відновлення одного вузла). Надалі вузол будемо називати заявкою. Якщо всі бригади зайняті, то заявка стає в чергу, K - число обслуговуючих апаратів, тобто

ремонтних бригад. На рисунку 2.8 схематично представлений процес відмов і відновлення вузлів кластера.

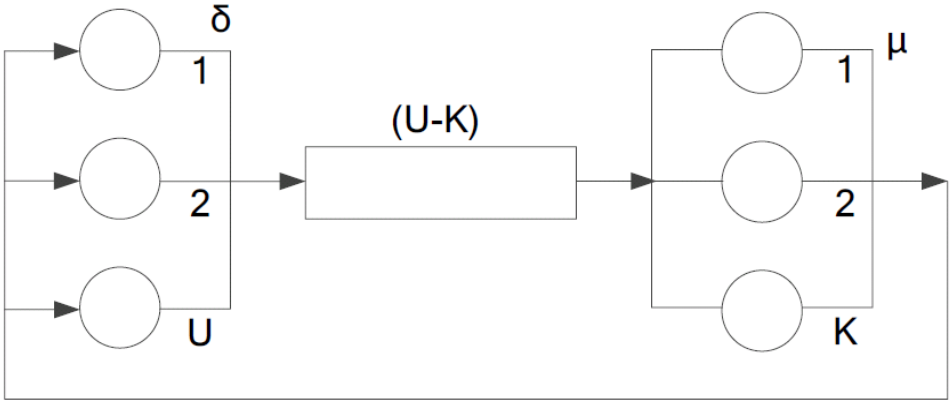


Рисунок 2.8 - Процес виходу з ладу і відновлення вузлів кластера

Нехай час напрацювання на відмову вузла і час його обслуговування (відновлення) розподілені за експоненціальним законом з параметром $1/\delta$ і μ. Тоді цей процес можна описати у вигляді системи масового обслуговування $M/M/K/U/U$. Імовірність p_i того, що в системі знаходиться i заявок, дорівнює:

$$P_i = \left\{ p_0 \cdot \left(\frac{\delta}{\mu}\right)^i \cdot C_U^i, \text{якщо} i < \text{K}; p_0 \cdot \left(\frac{\delta}{\mu}\right)^i \cdot C_U^i \cdot \frac{i!}{K!} \cdot K^{K-i}, \text{якщо} i \geq K \right\} \quad (2.4)$$

У розглянутій вище аналітичній моделі час відновлення вузла розподілено по експонентному закону. Але цей час може мати велику дисперсію. У цьому випадку необхідно використовувати модель $M/G/K/U/U$ з довільною функцією розподілу ймовірностей часу відновлення. Для такої моделі не існує аналітичного рішення. В цьому випадку необхідно використовувати імітаційну модель. Ця модель приведена в [14]. Для оцінки ймовірності p_0 був розроблений спеціальний прийом, який описаний нижче при розгляді структури імітаційної моделі на мові GPSS, представленої на рисунку 2.9.

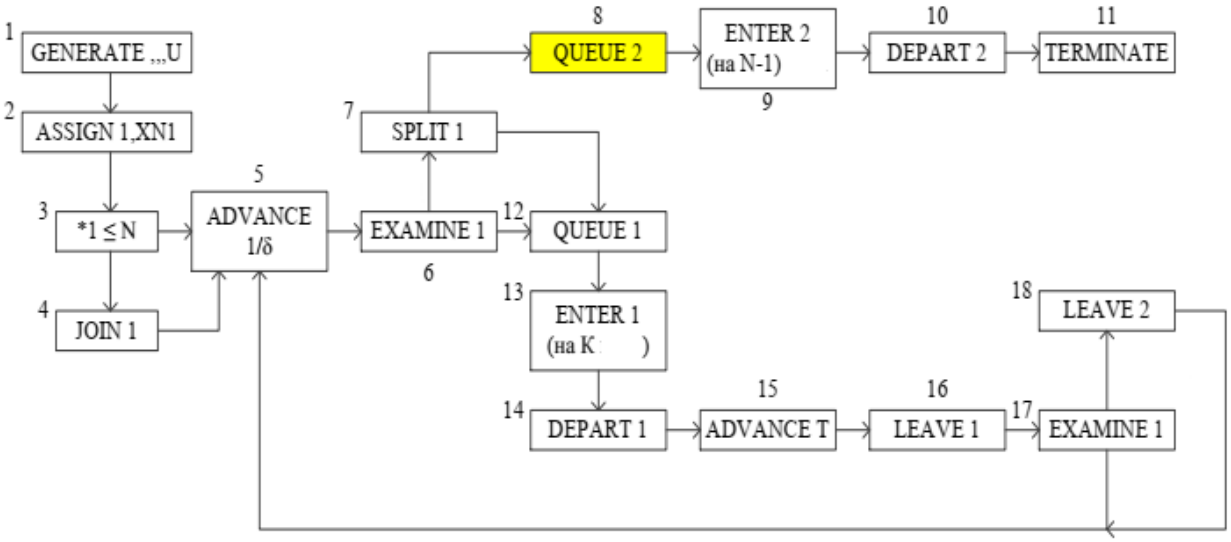


Рисунок 2.9 - Структура імітаційної моделі на мові GPSS

Генеруються U транзактів (вузлів) (див. мітку 1); в параметрі 1 транзакта зберігається номер транзакта (мітка 2); перші N тразактів об'єднуються в групу 1 (мітки 3,4). Група 1 - це вузли, де зберігаються N реплік будь-якого записи. Це перші N транзактів (в силу симетричності моделі номера вузлів не мають значення). Далі транзакти затримуються на час безвідмовної роботи (Експоненціальний розподіл ймовірностей із середнім $1/\delta$) (див. Мітку 5). Після відмови вузла визначається, чи належить транзакт групі 1 (мітка 6). Якщо «так» (відмовив вузол з реплікою), то генерується копія транзакта (мітка 7). Копія передається на вхід черзі 2 (мітка 8), потім на вхід пам'яті (мітка 9) (ємність пам'яті дорівнює N-1), далі черга 2 зменшується (мітка 10) та копія.

Копія транзакта (репліка запису) затримується в черзі 2 (мітка 8) тільки в тому випадку, якщо в системі вже присутні несправні вузли з N-1 репліками. В цьому випадку час очікування транзакта в цій черзі одночасного перебування всіх вузлів з N репліками в системі (відмовили всі вузли з репліками). Частка сумарного часу очікування від усього часу моделювання і є шукана ймовірність p_0. Ця частка дорівнює значенню, яке зберігається в Q_{a2} (середня довжина черги 2 - в цій черзі може бути або 0 транзактsв, або 1).

Транзакт групи 1 (вихідне повідомлення) або транзакт, що не належить групі, передається на вхід черзі 1 відмови вузлів (мітка 12). Далі він займає місце в

пам’яті 1 (мітка 13), якщо є вільне місце (це багатоканальний пристрій - його ємність дорівнює числу ремонтних бригад *K*). Потім черга 1 зменшується (мітка 14), тобто вузол приймається на обслуговування ремонтною бригадою (мітка 15). *T* - випадковий час відновлення вузла. Далі ремонтна бригада звільняється (мітка 16). Якщо транзакт не входить в групу досліджуваних реплік (мітка 17), то він повертається в блок з міткою 5. Якщо транзакт входить в групу 1, то перед цим він зменшує пам'ять 2 на 1 (мітка 18) - репліка відновлена. Якщо $N = 1$ (немає реплікації), то в імітаційної моделі треба прибрати чергу 2 (тобто блоки *QUEUE* 2 і *DEPART* 2 - див. рисунок 2.9) і встановити обсяг пам’яті 2, рівний 1. Тоді $p_0 = NA2$ (коефіцієнт використання пам'яті 2). Як функції розподілу ймовірностей часу відновлення вузла використовувалася наступна дискретна функція GPSS:

RECV FUNCTION RN3, D3 (2.5)

Аналіз результатів аналітичного моделювання показав, що при зміні середнього часу напрацювання на відмову існує час $T_{кр} = 1/\delta_{кр}$, менше якого p_0 зростає відразу на кілька порядків. На рисунку 2.10 показані залежності $T_{кр}$ від *U* при різних значеннях числа ремонтних бригад *K (N = 3)*.

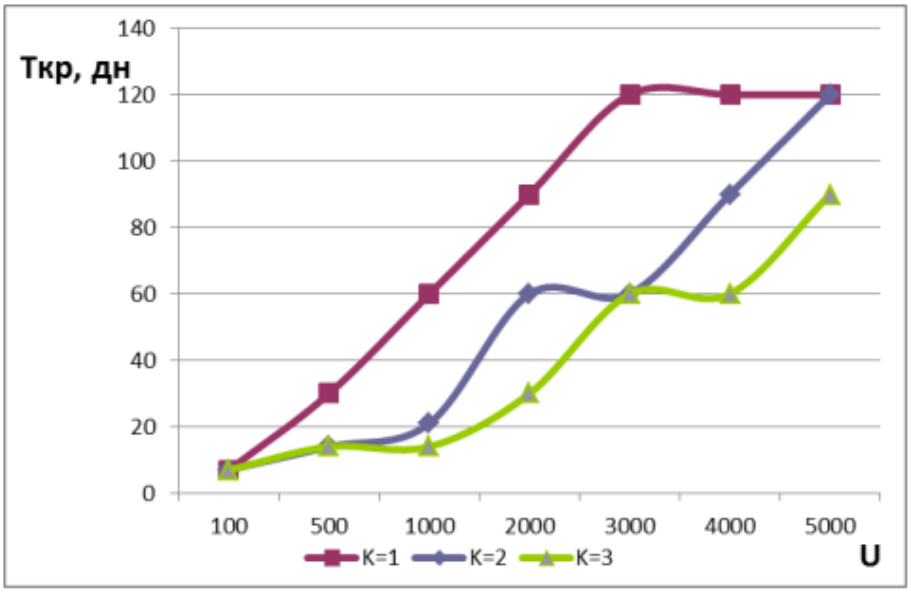


Рисунок 2.10 - Залежність часу $T_{кр}$ від числа вузлів *U* при різних значеннях *K*.

На рисунку 2.11 представлена залежність ймовірності p_0 від числа вузлів U при $N = 1$ (відсутня реплікація).

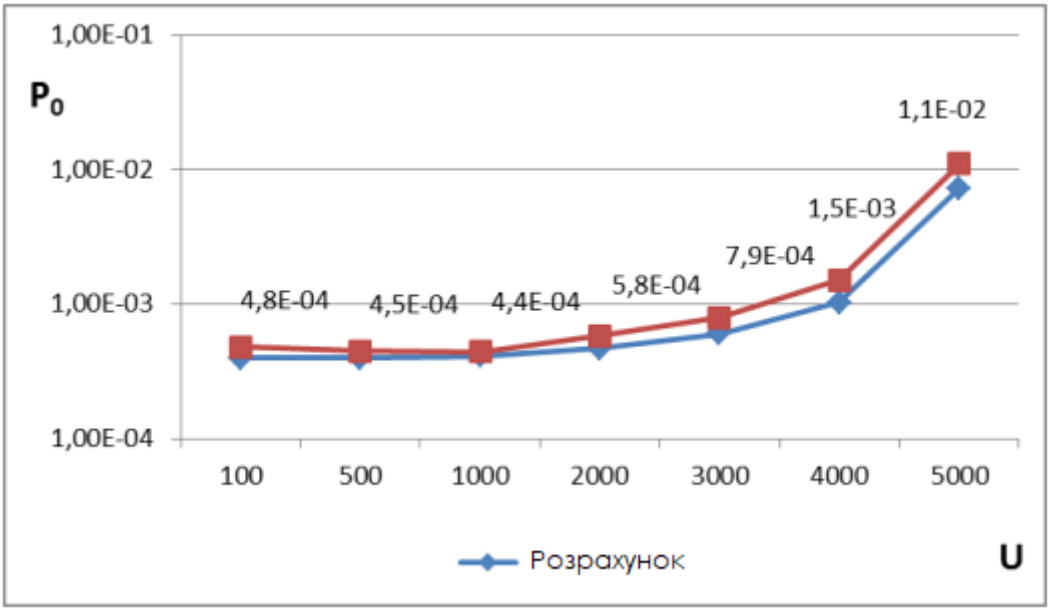


Рисунок 2.11 - Залежність ймовірності p_0 від числа вузлів U при $N = 1$ ($K = 2,1$ / $\delta = 3$ міс).

Слід зазначити, що графіки мають дуже пологу ділянку (і) при кожному K, тобто $T_{кр}$ практично не змінюється на цій ділянці.

2.4 Оцінка адекватності моделі управління реплікаціями в MongoDB

Для аналізу адекватності розроблених моделей процесів узгодження реплік і ведення версій записів була виконана серія натурних експериментів в хмарному кластері [13] розміром до 24 вузлів з використанням бази даних MongoDB [19].

Розглянемо деякі особливості розгортання комп'ютерного кластера в хмарному середовищі. На ринку існує велика кількість компаній, які надають хмарні обчислювальні ресурси. Ресурси бувають віртуальними і виділеними. Виділеними хмарними ресурсами є вузли, яким відповідають фізично окремі сервери. На відміну від виділених вузлів, віртуальні вузли - це віртуальні машини.

Оренда виділених серверів набагато дорожче, тому в нашому експерименті використовувалися віртуальні вузли, надані компанією DigitalOcean (DO) [23]. Оскільки досліджувані значення (ймовірність читання застарілого записи, характеристики часу очікування закінчення оновлення та ін.) оцінювалися при роботі користувача з одним записом <ключ / значення>, база даних NoSQL не навантажувала оперативну пам'ять, що дозволило орендувати недорогі віртуальні сервери з малим об’ємом оперативної пам’яті.

Всі віртуальні вузли, що надаються постачальником хмарних ресурсів, використовують ресурси багатопроцесорних машин з SSD дисками. Це дозволяє зробити припущення, що інші клієнти DO, яким надано віртуальні вузли на тому ж фізичному сервері, що не будуть навантажувати саме той процесор, на якому запущена наша віртуальна машина. Отже, продуктивність нашого віртуального вузла істотно не залежить від фонового завантаження процесора. При ініціалізації вузла є кілька опцій, серед яких можна виділити опцію private networking. При включенні даної опції всі вузли, орендовані користувачем, гарантовано находятся в одному центрі обробки даних (ЦОД), що означає відсутність підкластера мережі. Отже, параметр μ_{ns} - інтенсивність передачі даних по мережі, з'єднує підмережі - можна не враховувати.

При початковому налаштуванні вузла (Droplet) необхідно вибрати операційну систему (ОС) або образ, раніше створений користувачем. Використовувалася ОС Ubuntu Server, попередньо встановлена. Як система NoSQL була використана база даних MongoDB. Для установки і настройки MongoDB необхідно виконати ряд дій на кожному з вузлів кластера. Установка бази даних «з нуля» вимагає багато часу. Тому для прискорення підготовки кластеру загальна частина дій по установці системи, описаних в [19], було виконано один раз на одному вузлі, далі був зроблений образ вузла, який згодом був розтиражований на інші вузли.

Команда 1 виконувалася на кожному вузлі, крім одного, до якого приєднані інші вузли (ip_first_node). Після виконання команди 1 необхідно було перевірити конфігурацію кластера на будь-якому з вузлів командою 2, після чого зберегти

зміни командою 3. Після виконання всіх команд кластер був готовий до роботи, проте потрібно якийсь час для перенесення вже збережених даних з першого вузла на всі інші. Протягом цього часу система може відповідати not found на запити клієнтів.

Для проведення експериментів були розроблені прикладні програми для кожної з моделей. База даних MongoDB надає бібліотеки для доступу до системи на мовах Java, Erlang, Pyton, Ruby, з яких було обрано бібліотека Java. Java-додатки транслюються в проміжний байт-код, який виконується на будь-якій віртуальній машині, що полегшило налагодження додатку. Прикладні програми запускалися безпосередньо на хмарних вузлах, виробляли звернення до бази даних відповідно до алгоритму і накопичували статистику (журнали). Після проведення експериментів статистика передавалася на локальну машину для аналізу.

Аналізувався режим з параметрами реплікації $W = R = 1$. У моделі узгодження реплік в кінцевому рахунку вхідний потік вимог на читання до однієї репліки приймався пуассонівським з параметром λ. Для того, щоб експеримент відповідав моделі, вимоги на читання повинні надходити незалежно до кожної з *2* ... *N* реплік з інтенсивністю λ. До 1 вузла надходять вимоги на зміну запису з інтенсивністю 1.25 (1/сек).

Нижче наведені алгоритми програм, які виконуються на вузлах *1* ... *N* в процесі експерименту і при обробці журналів для оцінки ймовірності, що клієнт (процес читання на вузлі 2 ... *N*) прочитає застарілий запис за час розповсюдження оновлень записів по її *N-1* реплік.

Алгоритм процесу оновлення записів, що виконується на вузлі 1, який представлено на рисунку 2.12. Вхід: N_ITER_W - кількість ітерацій, KEY - ключ оновлюваного запису, REC_VAL - лічильник версії запису (значення).

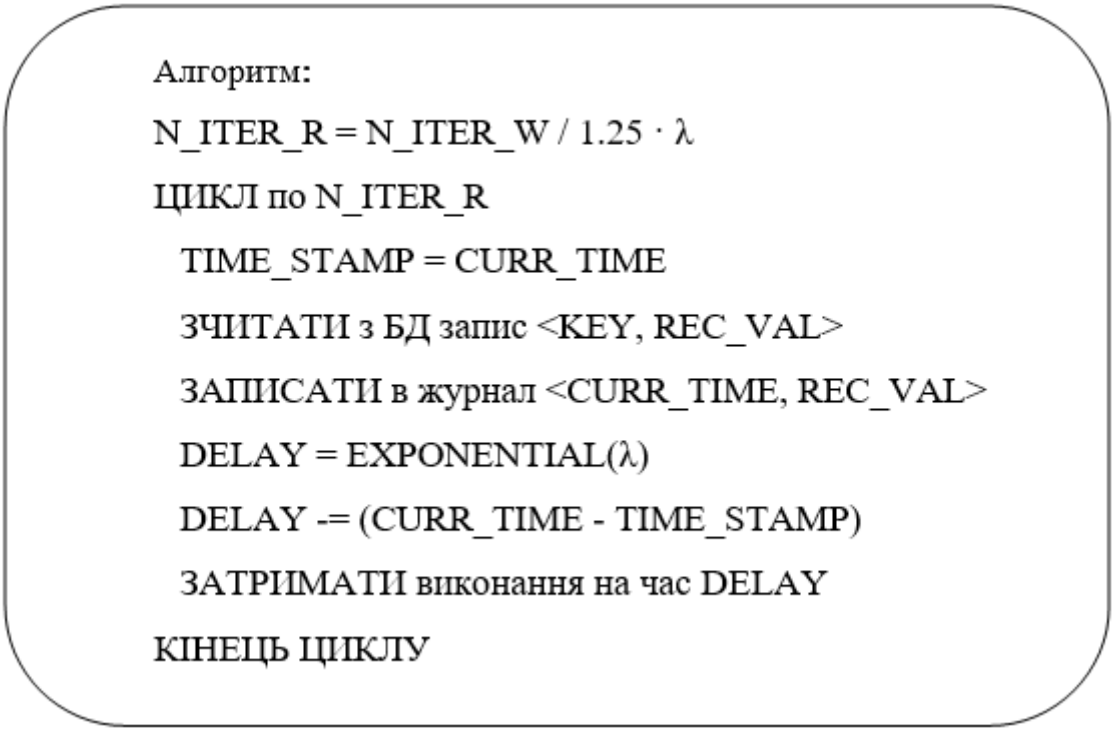


Рисунок 2.12 - Алгоритм процесу оновлення записів

Функція CURR_TIME повертає поточний час системи в мілісекундах. Зменшення DELAY компенсує час виконання алгоритму. При проведенні подібних експериментів в кластері виникає питання синхронізації часу між віртуальними вузлами. Цю проблему вирішує постачальник, встановлюючи єдиний час при ініціалізації вузлів.

Алгоритм процесів читання, виконуваних на вузлах *2 ... N*, який представлено на рисунку 2.13. Вхід: λ - інтенсивність надходження вимог на читання, KEY – ключ оновлюваного запису, N_ITER_R - число ітерацій процесу читання.

Алгоритм програми оцінки ймовірності, що клієнт (процес читання на вузлі 2 ... N) прочитає застарілу запис за час розповсюдження оновлень записів по її N-1 реплік представлено на рисунку 2.14. Збільшення змінної COUNT на одиницю розуміють так: після того, як завершена операція запису в 1-у репліку, надійшла вимога на читання і по ньому була прочитано застарілий запис.

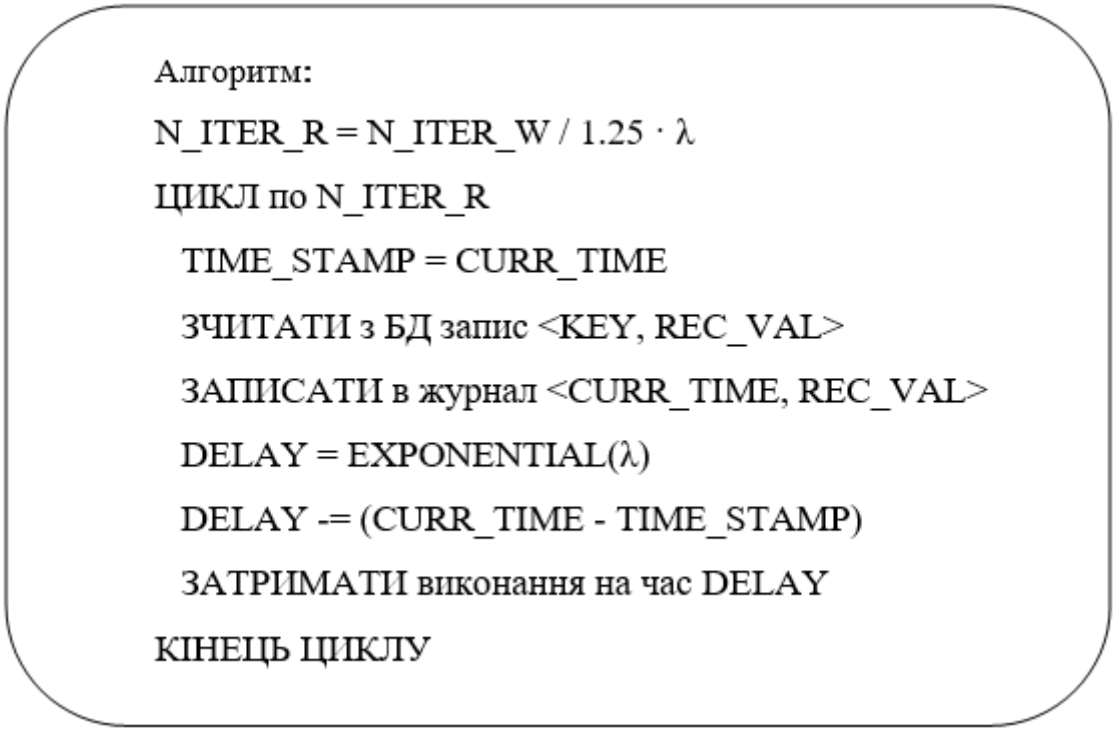


Рисунок 2.13 - Алгоритм процесу зчитування записів

Вхід: LOG_W - файл журналу процесу запису, LOGS_R - файли журналів процесів читання, N_ITER_W - число ітерацій процесу запису.

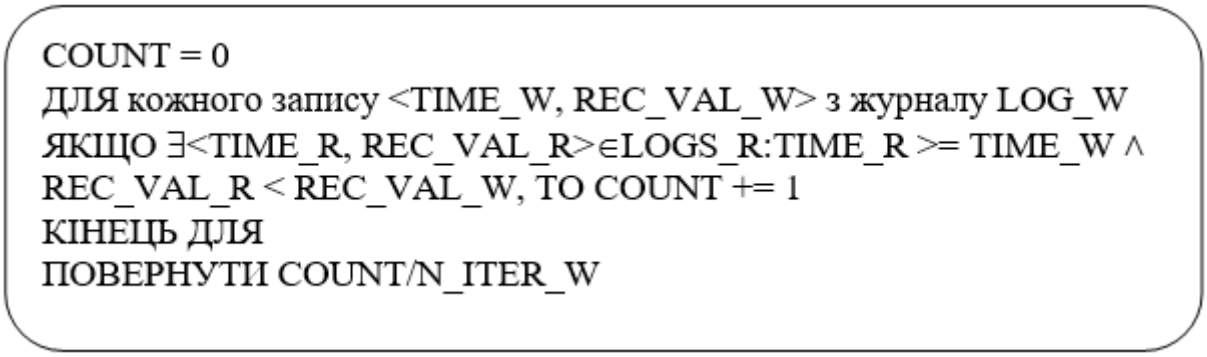


Рисунок 2.14 - Алгоритм процесу оцінки ймовірності оновлення записів

На рисунках 2.15 і 2.16 показані залежності ймовірності *P*, що клієнт прочитає застарілий запис, від λ при різних значеннях *N*. Графіки побудовані за результатами натурних (experiment) і модельних (model) експериментів.

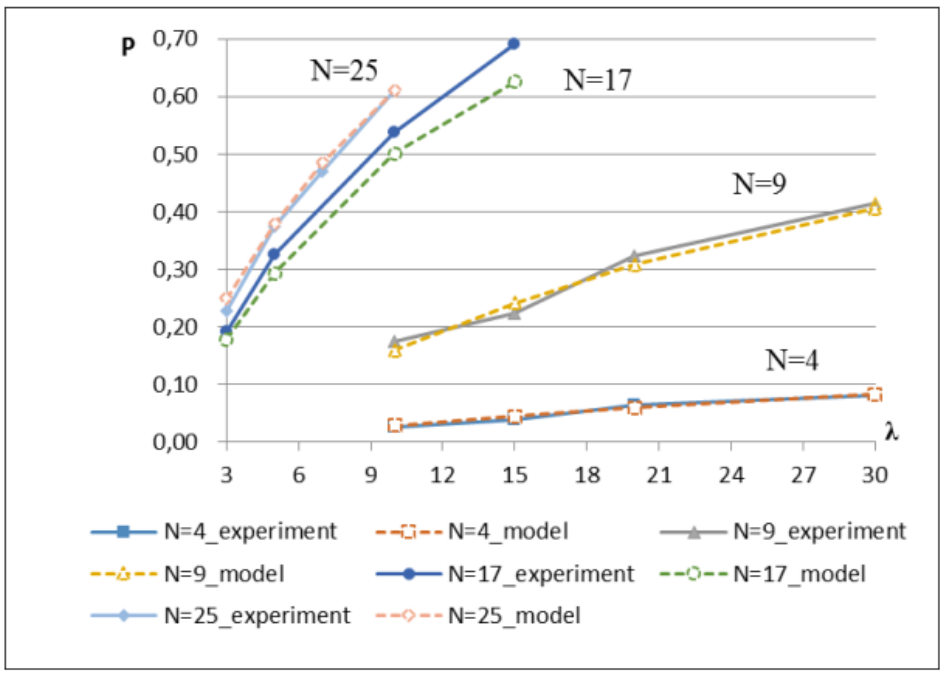


Рисунок 2.15 - Залежності ймовірності *P* читання застарілого запису від λ для першої серії експериментів.

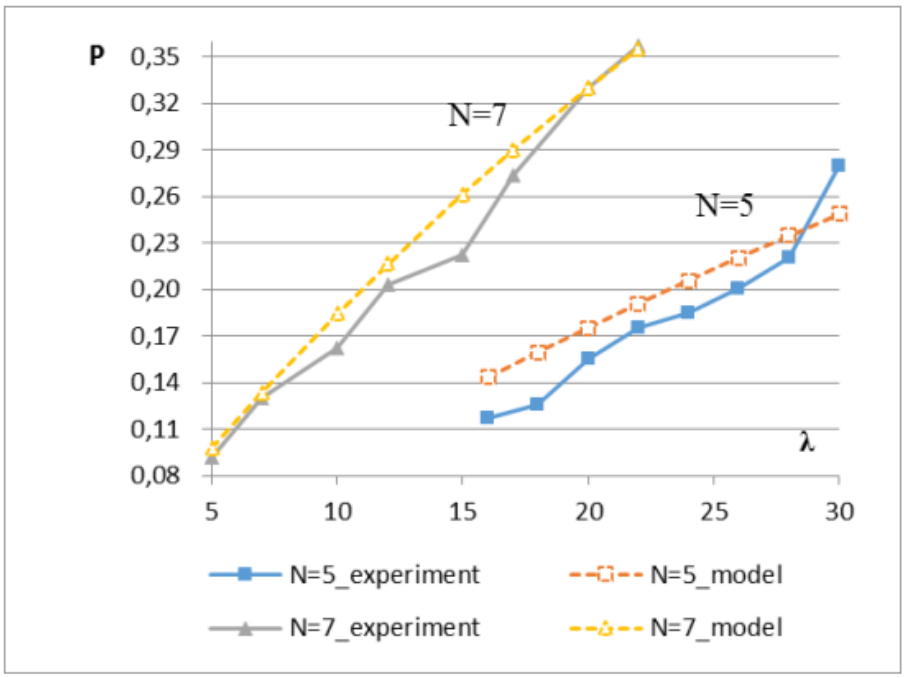


Рисунок 2.16 - Залежності ймовірності *P* читання застарілого запису від λ для другої серії експериментів.

Середня відносна похибка по кількох проведених серіях експериментів склала 7.86%.

В рамках виконання магістерського дослідження розроблено аналітичну модель процесу узгодження реплік в кінцевому рахунку в базах даних MongoDB, при цьому враховувався синхронний і асинхронний способи поширення змін. Модель дозволяє оцінити ймовірність того, що клієнт прочитає застарілий запис за час розповсюдження оновлень записів по її N-W реплік.

Розроблено аналітичну модель процесу суворого узгодження реплік в базах даних MongoDB. Модель дозволяє оцінити характеристики випадкового часу очікування вимог на читання закінчення оновлень W реплік, а також часу читання R записів з урахуванням цього очікування. Оскільки в якості серверної частини інформаційної системи управління HR-процесами використовується MongoDB, то зазначені моделі апробовані в рамках даної предметної області.

3 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ АВТОМАТИЗАЦІЇ HR-ПРОЦЕСІВ

3.1 Специфікація вимог до системи

Етап аналізу полягає в дослідженні системних вимог і проблеми. Зміст поняття аналізу більш точно відображають терміни аналіз вимог (requirment analysis) (тобто дослідження вимог до системи) і об'єктно-орієнтований аналіз (object-oriented analysis) (дослідження об'єктів предметної області). У процесі об'єктно-орієнтованого аналізу основна увага приділяється визначенню і опису об'єктів (або понять) в термінах предметної області.

Аналіз вимог може включати опис процесів або сценаріїв використання програми, яке може бути представлено у формі прецедентів. Об'єктно-орієнтований аналіз пов'язаний з описом предметної області з точки зору класифікації об'єктів. Декомпозиція предметної області завдання полягає в ідентифікації понять, атрибутів і асоціацій з предметної області, що мають важливе значення для вирішення завдання. Результат аналізу виражається в моделі предметної області (domain model), яка ілюструється за допомогою набору діаграм із зображеними на них поняттями або об'єктами предметної області.

Виділяти процеси найлогічніше буде, прив'язуючи процеси до існуючих структурних елементів. Існуючі елементи створювалися за функціональним принципом - виконання будь-якої функції, створення закінченого продукту. Таким чином, виділення процесів буде проводитися в рамках вже існуючої системи управління. Процеси предметної області можуть описуватися у формі прецедентів словесних описів в структурованому форматі. Прецедент - це набір взаємопов'язаних успішних і невдалих сценаріїв, що описує використання системи виконавцем для вирішення однієї з реквізит. На підставі запитів зацікавлених сторін [11] були сформовані описані надалі вимоги.

Функціональні. Функції з найменуванням "управляти" припускають наявність можливостей: додати, редагувати, видалити та ін. Функціональні вимоги виражені за допомогою діаграм варіантів використання [12], рисунок 3.1.

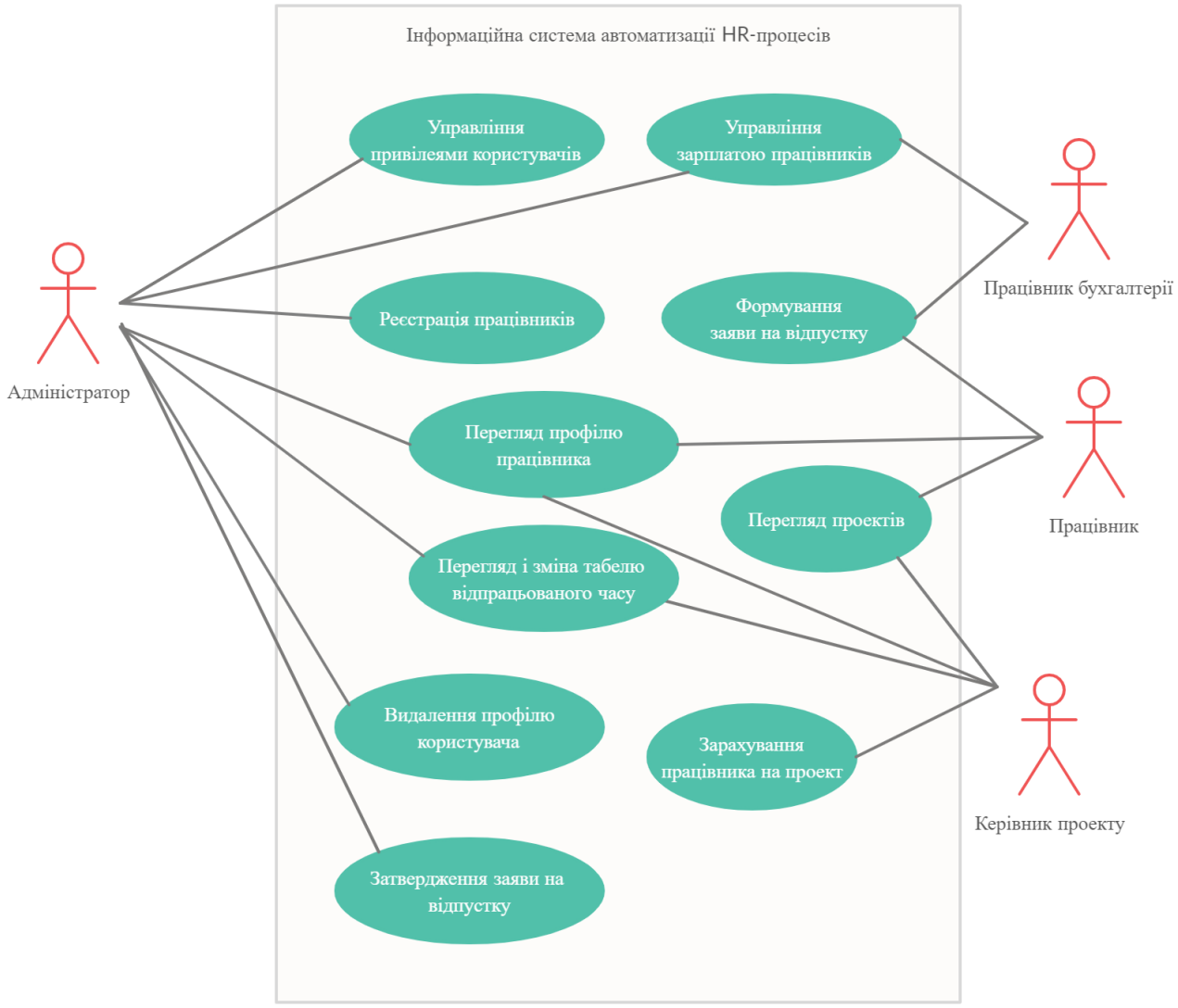


Рисунок 3.1 - Діаграма варіантів використання інформаційної системи автоматизації HR-процесів

Система зберігає інформацію про людські ресурси, освіту всіх співробітників, їх останній досвід, табель обліку робочого часу, управління відпустками та поточні розподілені проекти. Буде створена адміністративна панель для управління всією інформацією найнятих працівників. Він міститиме статистичні дані про коефіцієнт найму працівників з різних установ та середню

плинність працівників з різних установ. Цей проект буду побудований на Node Express, який є веб-додатком для Node.js, а інтерфейс створений за допомогою HTML, Bootstrap, CSS та JS.

Система формується з наступних основних структурних модулів:

- окремі облікові записи користувачів для всіх адміністраторів та працівників;

- різні представлення про дані та чіткий доступ до даних для учасників на основі їхніх привілеїв;

- реєстрація працівника/адміністратора;

- управління відпрацьованим часом всіх працівників;

- управління зарплатою всіх працівників.

- ведення обліку загального досвіду та практичного досвіду працівника.

- управління поточними розподіленими проектами працівника у межах організації.

В системі виділено наступні групи користувачів:

Адміністратор. Він має повний доступ до системи, яка включає реєстрацію працівників; прийняти рішення про привілеї для інших працівників, перегляд і зміна відвідуваності сьогоднішнього дня, перегляд і зміна заробітної плати працівника, видалення запису або профілю працівника, розподіл і перерозподіл проекту співробітнику, затвердження чи відхилення заяви про відпустку працівника.

Працівник. Він зможе позначити свій табель відпрацьованого часу, переглянути його історію, переглянути поточну заробітну плату, переглянути поточний профіль працівника (включаючи освітню та виробничу історію), переглянути всі свої проекти в межах організації, переглянути інших співробітників, які діляться з ним одним проектом, подати заяву на відпустку та переглянути статус заявок на відпустку.

Керівник проекту. Може ознайомитись із навичками співробітників, надати оцінку діяльності працівника, а також має доступ до працівників проекту, де він є керівником.

Працівник бухгалтерії: може сформувати відомість про оплату праці для кожного працівника, встановити премію для працівника, встановити заробітну плату працівнику, збільшити заробітну плату працівника, надіслати зарплатну відомість по електронній пошті кожному працівникові.

3.2 Розроблення архітектури інформаційної системи

Архітектура програми визначає його компоненти, їх функції та взаємодію. Розроблюваний додаток передбачає включення в себе клієнт-серверної взаємодії, ідентифікацію та авторизацію користувачів та зберігання даних.

Додаток повинен здійснювати передачу запитів по мережі. Архітектура програми повинна включати в себе клієнтську і серверну частину. Робота клієнт-серверного додатка повинна функціонувати таким чином:

- клієнт формує і посилає запит на сервер;
- сервер робить необхідні маніпуляції з даними і відправляє запит до бази даних;
- сервер отримує результат запиту з бази даних, формує результат і передає його клієнту;
- клієнт отримує результат, відображає його на пристрої виведення і чекає подальших дій користувача.

Цикл повторюється, поки користувач не закінчить роботу з сервером. Для вирішення даної бізнес проблеми, була вибрана клієнт-серверна архітектура (рисунок 3.2). Вона найбільш придатна для реквізит саме такого типу, коли потрібно забезпечити доступ до системи багатьом користувачам. Найголовнішим є те, що клієнт-серверна архітектура надає можливість віддаленого доступу [2].

Проект, який побудований на клієнт-серверній архітектурі, повинен складатися з трьох частин: зв’язок з базою даних, відображення даних клієнту та бізнес логіка проекту яка обробляє запити користувача і відображає саме ту інформацію, яку захотів користувач. Обробка та збереження даних відбувається на

боці сервера, відображення даних і надсилання запитів на їхню модифікацію виконується на боці клієнта.

У процесі проектування основна увага приділяється концептуальному рішенню (у вигляді програмного забезпечення або апаратних засобів), що забезпечує виконання основних вимог. Наприклад, на етапі проектування описуються програмні об’єкти або схеме бази даних.

Поняття проектування можна розділити на об'єктно-орієнтоване проектування (object-oriented design) і проектування бази даних (database design). У процесі об’єктно-орієнтованого проектування визначаються програмні об'єкти і способи їх взаємодії з метою виконання системних вимог.

Об’єктно-орієнтоване проектування пов’язане з визначенням програмних об’єктів, їх обов’язків і способів взаємодії. Для ілюстрації взаємозв’язків між об’єктами використовується діаграма послідовностей (sequence diagram), яка являє собою один з видів діаграми взаємодії UML. Вона відображає потоки повідомлень між програмними об’єктами і виклики методів. Крім динамічного Представлення взаємозв’язку об'єктів, що відображається на діаграмі взаємодій, дуже корисно будувати зріз системи у вигляді діаграми класів проектування.

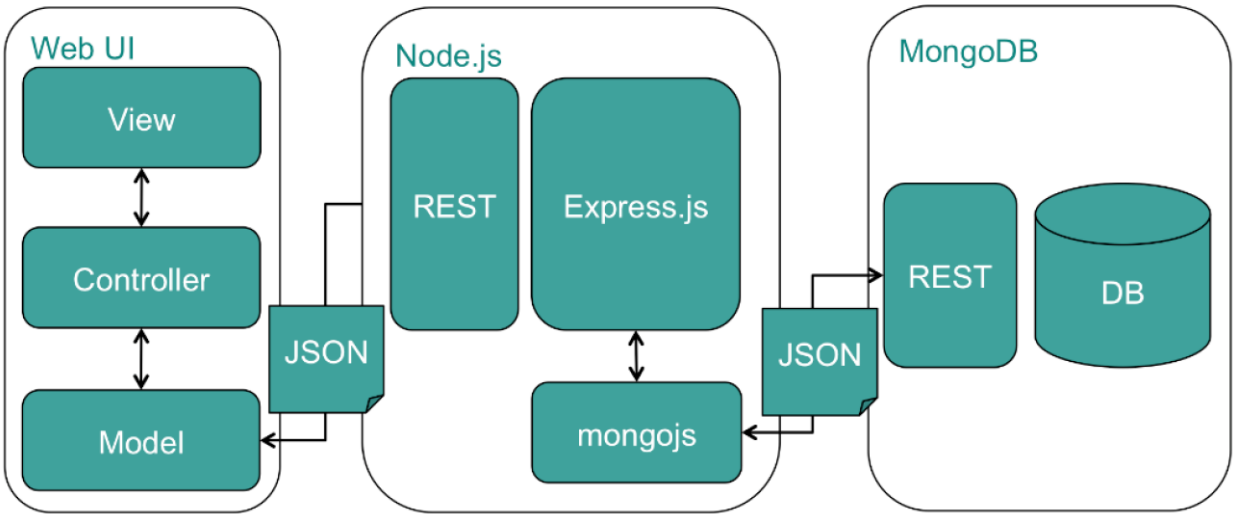


Рисунок 3.2 - Діаграма розгортання системи

Діаграми кооперації також як і діаграми послідовності відносяться до діаграм взаємодії. Вони ілюструють взаємодію об'єктів у форматі графа або мережі, відображають потік подій, загострюють увагу на зв'язках між об'єктами.

Моделювання реалізації операції здійснюється наступним чином:

1. Ідентифікація параметрів, повертається значення та інші об’єкти, видимі для операції.

2. Якщо операція тривіальна, уявіть її реалізацію безпосередньо в коді, який можна помістити на задній план моделі або явно візуалізувати в примітці.

3. Якщо операція алгоритмічно складна, змоделюємо її реалізацію за допомогою діаграми діяльності.

4. Якщо операція вимагає великого обсягу детального проектування, уявіть її реалізацію у вигляді кооперації. Надалі можна розгорнути структурну і поведінкову складові кооперації з допомогою діаграм класів і взаємодії відповідно. Зв'язок є з'єднанням між двома екземплярами класів, які визначають деяку форму переміщення і видимості між ними. Зв’язок-екземпляр асоціації. Передані між об'єктами повідомлення представляються у вигляді імен цих повідомлень над лініями зв’язків, позначених стрілками. Над однією лінією зв’язку може бути зазначено будь-яку кількість повідомлень. Для відображення порядку проходження повідомлень у поточному потоці управління поряд з повідомленням наводиться порядковий номер.

Діаграми кооперації, також як і діаграми послідовності можна використовувати для відображення взаємодії, як між об'єктами предметної області, так і між програмними об'єктами.

Існують різні способи організації взаємодії між клієнтом і сервером. Одними з найбільш популярних технологій є WebSocket і AJAX. Суть технології AJAX полягає в зміні вмісту завантаженої веб-сторінки без її повного перезавантаження, завдяки чому досягається висока динамічність сайтів. Технологія грунтується на поділі даних і підзавантаження тих чи інших компонентів у міру необхідності. AJAX з'явилася в 1998 і тому підтримується старими версіями різних браузерів [9].

WebSocket - технологія стандарту HTML5, що дозволяє встановлювати повнодуплексне TCP-з’єднання між сервером і веб-браузером (клієнтом). Ця технологія призначена для вирішення завдання зняття обмежень обміну даними між браузером і сервером. У браузері обов’язково повинна бути підтримка WebSocket.

Для розгляду деяких передумов вибору технічних або інших інструментів і технологій, а також проектування додатку, була розроблена діаграма аналізу. Уточним деякі варіанти використання згідно з діаграмою. На рисунку 3.3 представлені класи аналізу, які представляють деталізацію варіації використання «Реєстрація працівників».

Рисунок 3.3 - Діаграма аналізу «Реєстрація працівників»

На основі структури класів аналізу видно, що система зберігання даних є невід'ємною частиною додатку і може надавати дані, не залежно від призначеного для користувача інтерфейсу. А також на підставі того, що мають місце бути класи суті, управління і граничні, має сенс орієнтуватися на три відповідальних компонента: за представлення даних, управління та зберігання. В цьому випадку, традиційно використовується типове архітектурне рішення MVC.

При виборі засобів реалізації, відзначимо такі особливості:

- очікується тенденція до зростання кількості запитів на одиницю часу.
- один запит не вимагає "великих" обчислювальних ресурсів сервера.

В силу даних особливостей переважно орієнтуватися на засоби по обробці запитів або виконання коду програми, які мають асинхронну модель роботи, засновану на подіях, або надають такі механізми. Це обумовлювалося тим, що

асинхронна модель умовно дозволяє скоротити час відгуку, з тієї причини, що вона не блокує вхід-вихід і операційні завдання можуть виконуватися паралельно, підвищуючи загальну продуктивність системи на кожному рівні.

Умовне порівняння швидкості виконання коду програми при синхронній і асинхронної моделі, показані на рисунках 3.4 і 3.5 відповідно.

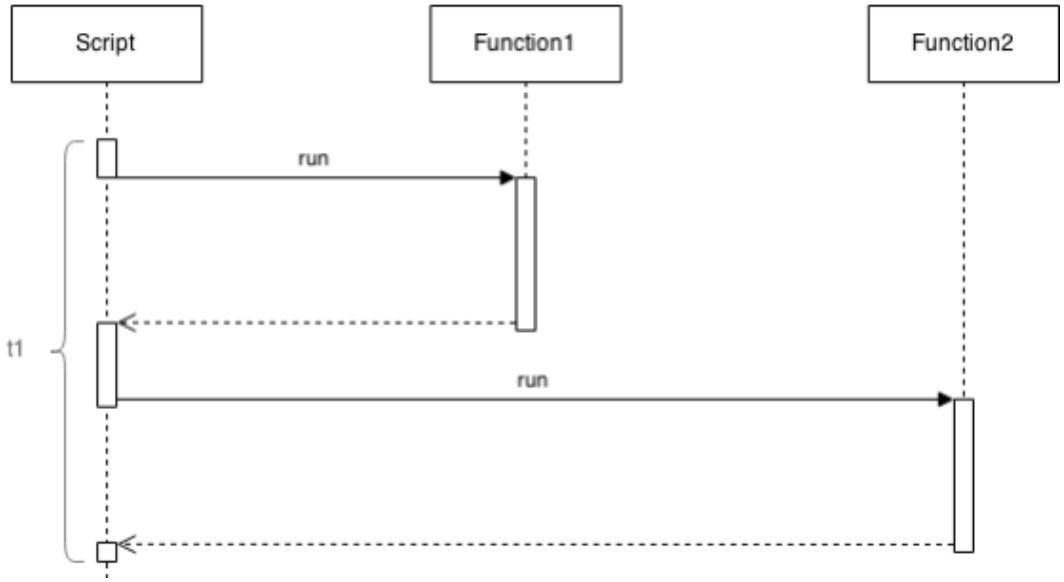


Рисунок 3.4 - Синхронна модель обробки запитів

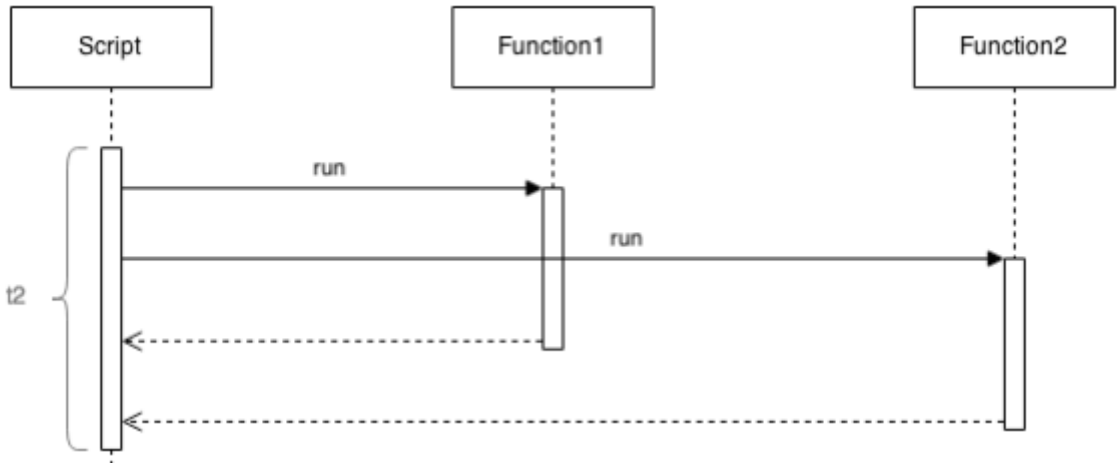


Рисунок 3.5 - Асинхронна модель обробки запитів

З рисунків видно, що t1> t2. Це говорить про те, що час обробки запиту або виконання коду при асинхронній моделі менше. Найчастіше, на рисунках, асинхронна модель передбачає демонстрацію функцій зворотного виклику.

В результаті проектування в цілому, систему можна розділити на компоненти, які представлено на рисунку 3.6.

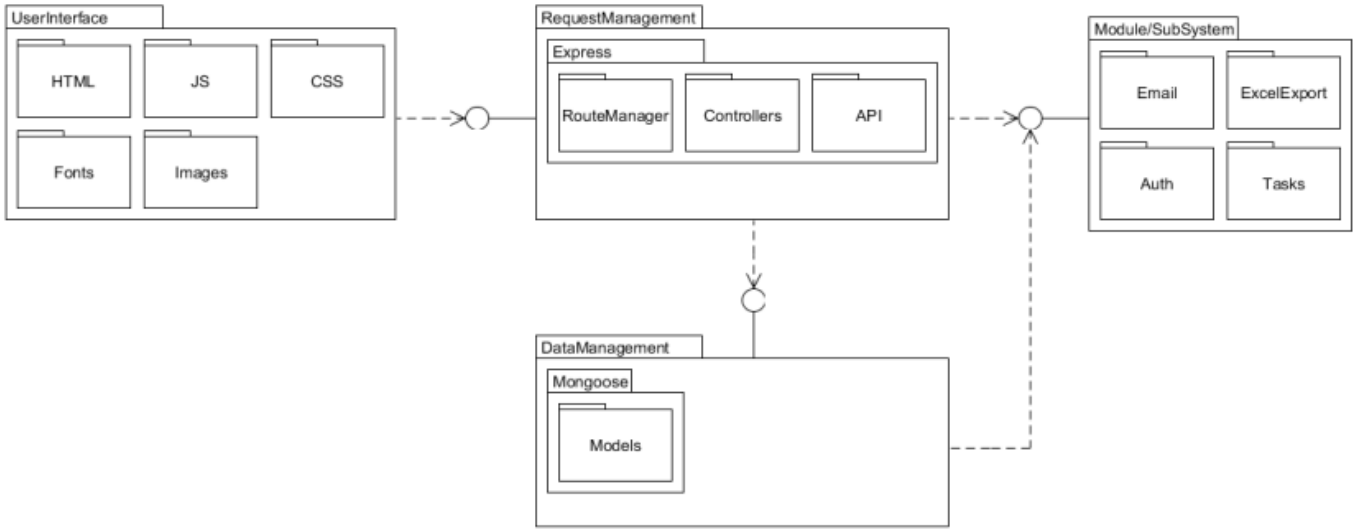


Рисунок 3.6 – Структурні компоненти системи автоматизації HR-процесів

Як уже згадувалося система використовує трирівневу архітектуру. Комунікація клієнтського додатка, сервера додатків і СУБД, здійснюється текстовими структурованими даними в форматі JSON, рисунок 3.7. Логіка клієнтської і серверної частини реалізуються на мові JavaScript.

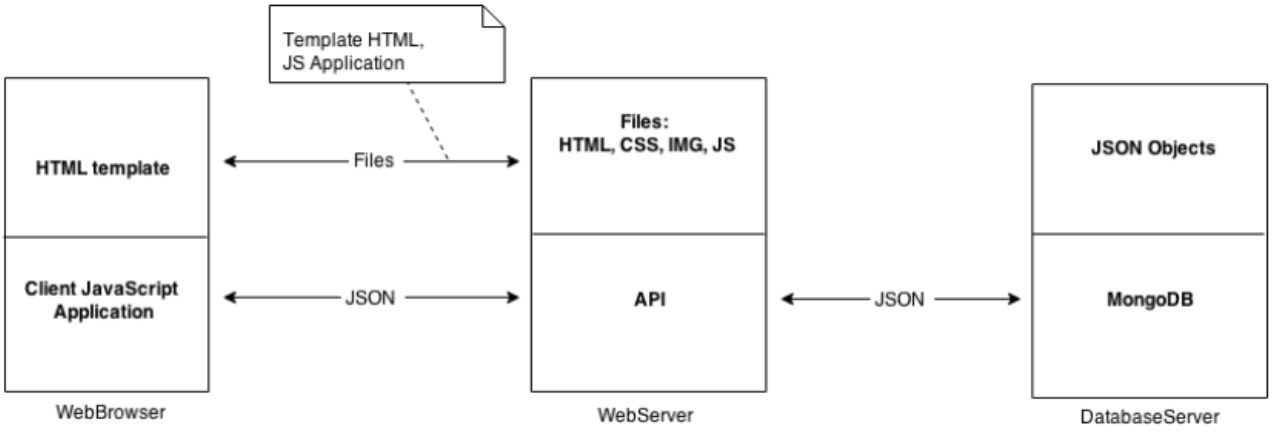


Рисунок 3.7 - Обмін даними в форматі JSON в трирівневій архітектурі

У цьому розділі розроблено архітектуру інформаційної системи, що дозволить краще зрозуміти функції основних її частин. Створено та описано структурну схему, основними компонентами якої є: рівень клієнта, рівень бізнес-логіки та рівень даних. Описано функціональну структуру системи та її основних елементів – модулів обробки даних.

4 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ HR-ПРОЦЕСІВ НА БАЗІ NODE.JS

4.1 Опис технологій для створення системи

Аналіз сучасних веб-технологій показав, що для побудови системи підходить платформа Node.js. Node.js — це JavaScript-оточення побудоване на JavaScript-рушієві Chrome V8. Node.js використовує подієву, неблокуючу I/O модель, що робить його легким та ефективним [21].

Прийнята в Node.js (рисунок 4.1) модель принципово відрізняється від поширених платформ для побудови серверів, в яких масштабованість досягається за рахунок багатопотоковості. Завдяки подійно-орієнтованій архітектурі знижується споживання пам’яті, підвищується пропускна здатність [22, с.16].

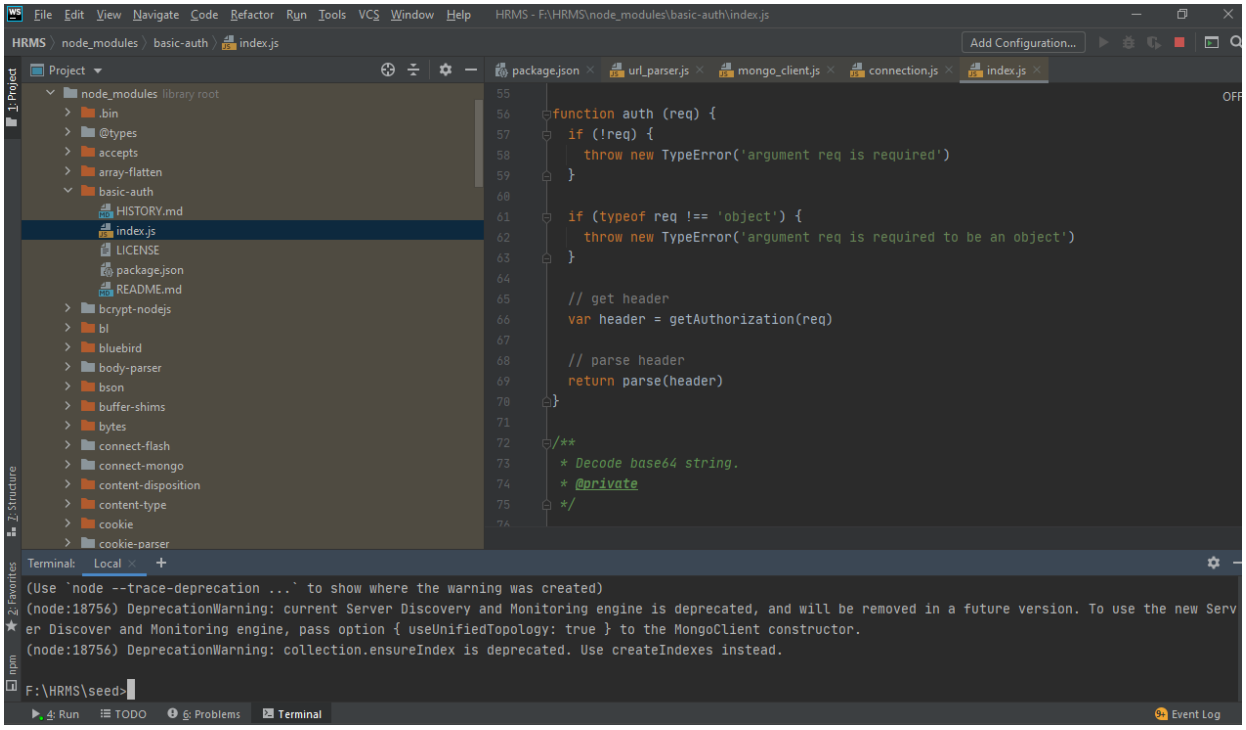


Рисунок 4.1 - Середовище розробки з Node.js

У Node.js при старті створюється єдиний програмний потік. Node.js виконується в цьому потоці. Це означає, що дві або більше частини додатка одночасно не можуть виконуватися [3]. Можливість “одночасного” виконання коду

надається за допомогою використання асинхронної моделі, побудованої на черзі подій (event-loop). На рисунку 4.2. зображена діаграма розгортання.

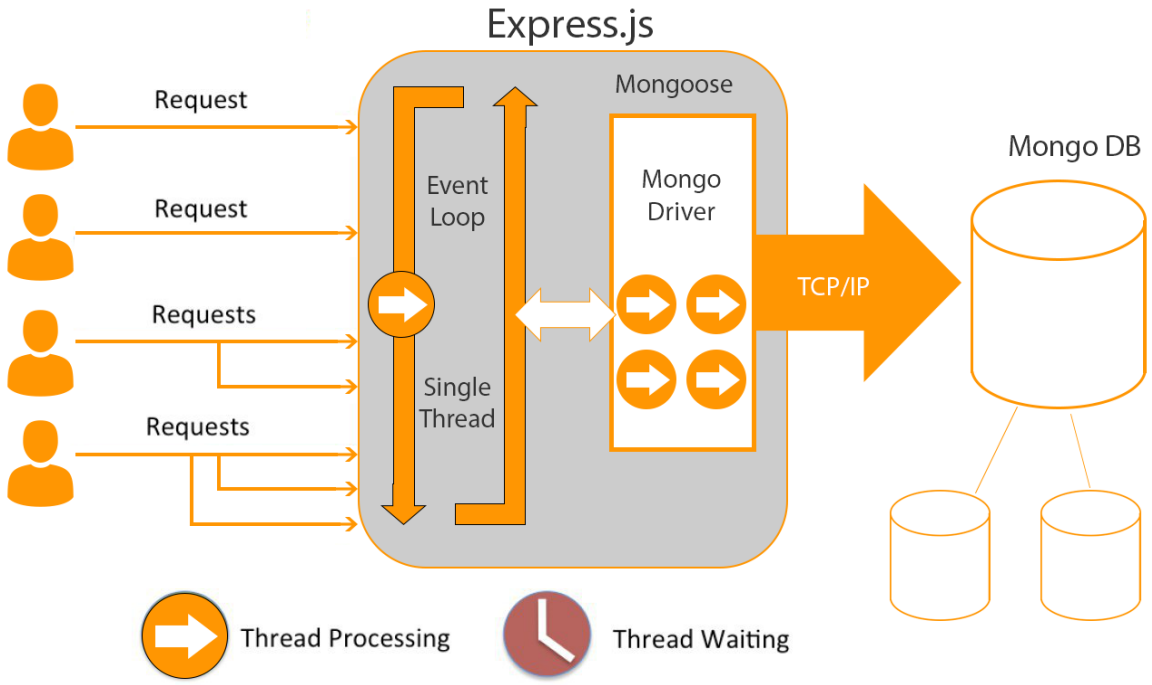


Рисунок 4.2 - Діаграма розгортання, включаючи Node.js Express, MongoDB

В якості СУБД було обрано базу даних MongoDB за наступними причинами:

- можливість опрацювання неструктурованих даних;
- відсутність або часткова зв'язність між колекціями;
- збереження даних не вимагає високих показників.

Кожен екземпляр знаходиться в базі даних у впорядкованій колекції (рисунок 4.3).

Загальна архітектура веб-додатку. В основі розроблювальної системи лежить клієнт-серверна архітектура. Клієнт робить запит до сервера на виконання певних дій або представлення деякої інформації, сервер отримує запит, виконує його та відправляє відповідь.

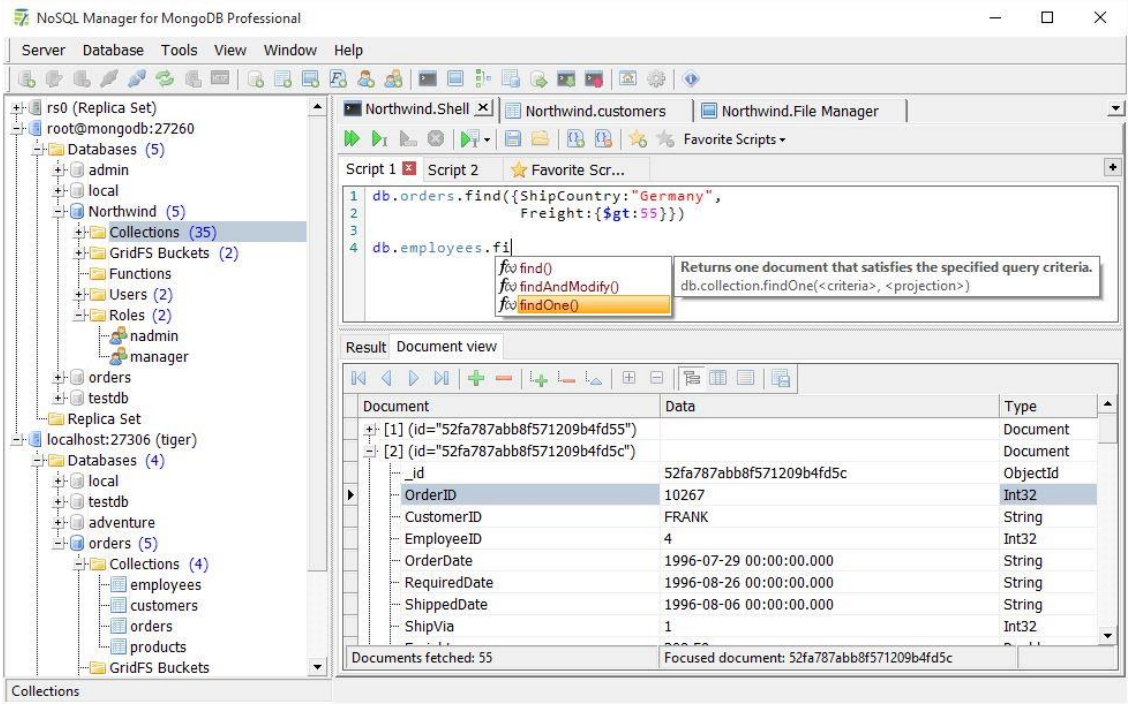


Рисунок 4.3 - MongoDB Manager

Серверна частина системи управління HR-процесами для обробки запитів повинна мати три основні компоненти, які будуть відповідати за: обробку запиту, перед обробку надісланих даних та обробку маршруту. У компоненті, відповідальному за обробку запиту, створюється об’єкт даного запиту та передається до компоненту відповідального за обробку надісланих даних, після чого екземпляр, створений для обробки запиту, звільняється.

Функції які відповідають проміжну обробку надісланих даних, у фреймворку ”Express”, називаються “middleware”. Вони об’єднуються в ланцюжки та після виконання всіх “middleware” певного ланцюжка, управління передається до компоненту, відповідального за обробку маршруту. Звільнення екземпляру обробки запитів та подальше передання управління до компоненту, відповідального за обробку маршруту, виконується за допомогою функції зворотного виклику.

Побудова серверної частини за допомогою програмної платформи Node.js та використання документо-орієнтованої бази даних MongoDB дає можливість створювати швидкісні додатки, орієнтовані на велике число одночасних з’єднань, вимагаючи при цьому невелику кількість оперативної пам’яті.

4.2 Програмна реалізація системи

Для розробки проекту було вирішено використовувати мову документа JavaScript з платформою Node.js. Це безкоштовна платформа, яка дозволяє писати програми на JavaScript, і поставляється з множиною корисних модулів, так що немає необхідності писати код «з нуля».

Node.js складається з середовища виконання і бібліотек. При створенні програми були використані наступні JavaScript бібліотеки:

- express.js - фреймворк для зручного створення веб-додатків [12];

- socket.io - JavaScript бібліотека для реалізації Web-Socket з'єднання в веб-додатках та обміну даними в реальному часі. Складається з двох частин: клієнтської, яка запускається в браузері і серверної для node.js. Обидва компоненти мають схожу API [14];

- WebStorm - самостійний редактор коду з підсвічуванням синтаксису, темами, і гарячими клавішами, написаний на JavaScript код легко вбудовується в будь-яку веб-сторінку [10];

- monk.js - обгортка для бази даних MongoDB, що дозволяє швидко та зручно взаємодіяти з колекціями бази [13].

Для реалізації обміну повідомленнями між клієнтом і веб-сервером в режимі реального часу був обраний WebSocket. При такому з'єднанні обидві частини (клієнтська і серверна) рівноправні і мають можливість перевіряти зв'язок за допомогою керівних фреймів типу PING і PONG. Той, хто хоче перевірити з'єднання, відправляє кадр PING з довільним тілом. Його одержувач повинен відповісти фреймом PONG з тим же тілом за розумний час.

Для обміну даними між клієнтською і серверною стороною були спроектовані і реалізовані обробники подій. Список обробників подій на стороні клієнта представлений на рисунку 4.4.

```
 socket
.on('message', function (username, message) {/*…*/})
.on('leave', function(username, userList) {/*…*/})
.on('join', function(username, userList) {/*…*/})
.on('connect', function() {/*…*/})
.on('disconnect', function() {/*…*/})
.on('change code', function(code){ /*…*/})
.on('change lang, function(lang) {/*…*/})
.on('change rights', function(readOnly){ /*…*/});
```

Рисунок 4.4 - Список обробників подій на стороні клієнта

Детальніше обробники подій на стороні клієнта можна описати наступним чином:

- message - подія під час надсилання повідомлення в чаті веб-додатку;

- leave - подія при закритті всіх вкладок браузера учасника, вихід їх документа;

- join - подія при підключенні учасника до системи;

- connect - подія під час активного з’єднання з сервером;

- disconnect - подія при втраті з’єднання з сервером;

- change - подія при зміні доступу до проекту;

- change lang - подія при зміні мови;

- change rights - подія при зміні прав для конкретного учасника.

Також були спроектовані і реалізовані обробники подій на стороні сервера, які представлені на рисунку 4.5.

Потрібно розуміти, що хоча назви деяких подій на стороні клієнта і на стороні сервера збігаються, реалізація подій принципово відрізняється. Так, наприклад, при обробці події message на стороні клієнта слід вивести в чат повідомлення, отримане від сервера, тоді як сервер обробляє цю подію, відправляючи повідомлення всім учасникам чату.

На етапі проектування було вирішено використовувати документо-орієнтовану NoSQL СУБД. Існує кілька таких систем: CouchDB, Couchbase, MarkLogic, mongoDB, eXist і ін. При цьому важливо, щоб обрана система забезпечувала зручне взаємодія з реалізованим веб-додатком.

```
 socket
.on('add user', function(user, cb) {/*…*/})
.on('message', function(text, cb) {/*…*/})
.on('disconnect', function(username, cb) {/*…*/})
.on('change code', function(code, cb) {/*…*/})
.on('change lang, function(lang) {/*…*/})
.on('change rights', function(userID, task, readOnly) {/*…*/}
```

Рисунок 4.5 - Список обробників подій на стороні сервера

З наведених СУБД MongoDB є найбільш популярною для обраної платформи. Це СУБД з відкритим програмним забезпеченням, розширювана, високопродуктивна, вільна від схем, яка зберігає всі дані в форматі бінарного JSON (BSON). Головна відмінність MongoDB - гнучке використання в веб-додатках. Для зберігання даних по створених користувачами документах були спроектовані колекції з полями, представленими на рисунку 4.6.

Колекція містить ID користувача, який створив реквізиту, ID проектів, поточне отримання реквізитів платіжних відомостей та ID запису в коллекції. Було вирішено сприймати нову вкладку документа, як вкладку того ж користувача і відключати з’єднання з клієнтом тільки коли закриті всі вкладки, відкриті цим документом.

Складність полягає в тому, що за стандартом WebSocket нова вкладка браузера є повноцінним з’єднанням клієнта і сервера. Тому була створена таблиця підключених користувачів до конкретного проекту, щоб відстежити всі підключені користувачем сокети. Модель даних, що зберігаються представлена в таблиці 4.2.

```
{
        "_id" : ObjectId("5fcbca1e13dbcf3e443ebe04"),
        "Skills" : [
                "PHP",
                "Big Data Analytics"
        ],
        "email" : "gnativ@gmail.com",
        "type" : "employee",
        "password" : "$2a$05$OKrITwVdqnqFbvzVoVYIve41rcnSlfJeuMuQGCc7CXhgI29rNeev.",
        "name" : "Гнатів Оксана",
        "dateOfBirth" : ISODate("2020-12-02T00:00:00Z"),
        "contactNumber" : "0300-4814710",
        "department" : "Software Development",
        "designation" : "System Analyst",
        "dateAdded" : ISODate("2020-12-05T17:57:50.514Z"),
        "__v" : 0
}
{
        "_id" : ObjectId("5fcbd91313dbcf3e443ebe08"),
        "Skills" : [
                "Не визначено"
        ],
        "email" : "buhgalter@gmail.com",
        "type" : "accounts_manager",
        "password" : "$2a$05$JWTZcFhxjXEFsVn5rc74v.98MOUjDn.S45nTuqkCaauUy53P5DB02",
        "name" : "Крайняк Людмила",
        "dateOfBirth" : ISODate("1965-12-09T00:00:00Z"),
        "contactNumber" : "0300-4814710",
        "department" : "Accounts",
        "designation" : "Accounts Manager",
        "dateAdded" : ISODate("2020-12-05T19:01:39.059Z"),
        "__v" : 0
}
```

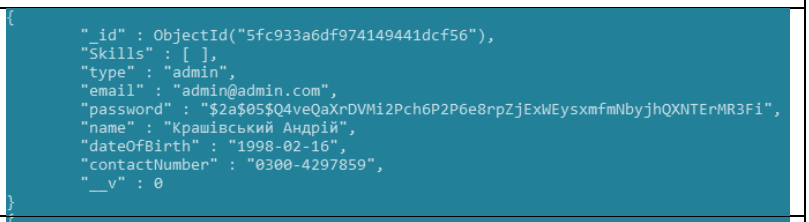
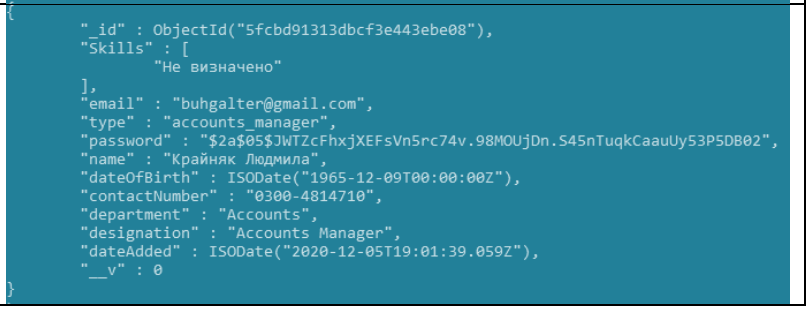

Рисунок 4.6 - Модель документів бази MongoDB

У MongoDB база даних складається з колекцій. Як правило, колекція відображає сутність предметної області. Колекція складається з "документів" (далі запис). Запис є JSON-об’єкт з обов’язковою наявністю ідентифікатора ObjectId, який MongoDB задає автоматично і контролює його унікальність, аж до унікальності в усьому світі [12]. ObjectId є типом даних. У порівнянні з реляційними СУБД, колекція є таблицею, а запис є рядком в таблиці.

Тип збережених в запису даних - довільний, з перерахованих нижче: String; Array (структура даних з доступом по цілочисельному індексу); Boolean; Date; Integer; Null. Object (структура даних, що надає доступ до елементів по ключу).

Цілісність зв'язків в MongoDB не підтримується. Це завдання вирішується на рівні коду сервера додатків, що і було реалізовано за допомогою інструменту ODM, - Mongoose.

Таблиця 4.2 - Приклад списку формування списку працівників

Ідентифікатор користувача	Роль	Реалізований Json-документ
5fc933a6df97 4149441dcf56	Адміністратор	
5fcbd91313dbc f3e443ebe08	Бухгалтер	

Для того щоб мати представлення відношення між сутностями, була розроблена схема БД, на якій відображені псевдо-зв’язки, рисунок 4.7, нотація Гордона Евересту [21].

Доповнимо до схеми, що найменування колекцій і псевдо-зв'язки можуть представляти дещо інший сенс, ніж таблиці в реляційних БД, так як існують вкладені об’єкти. Прикладом є колекція з найменуванням product_type.

Типи псевдо-відношень документа з іншими сутностями, враховуючи не структурованість даних:

- HasOne: має один тип документа.
- HasOne: має одну мову.
- ManyToMany: належить до одного або множини продуктів (BelongsTo), так як є загальним. З іншого боку один товар має множину документів на різних мовах (HasMany).
- HasMany: має багато змін.
- BelongsTo: належить до одного скачування, відправлення по email або замовлення.

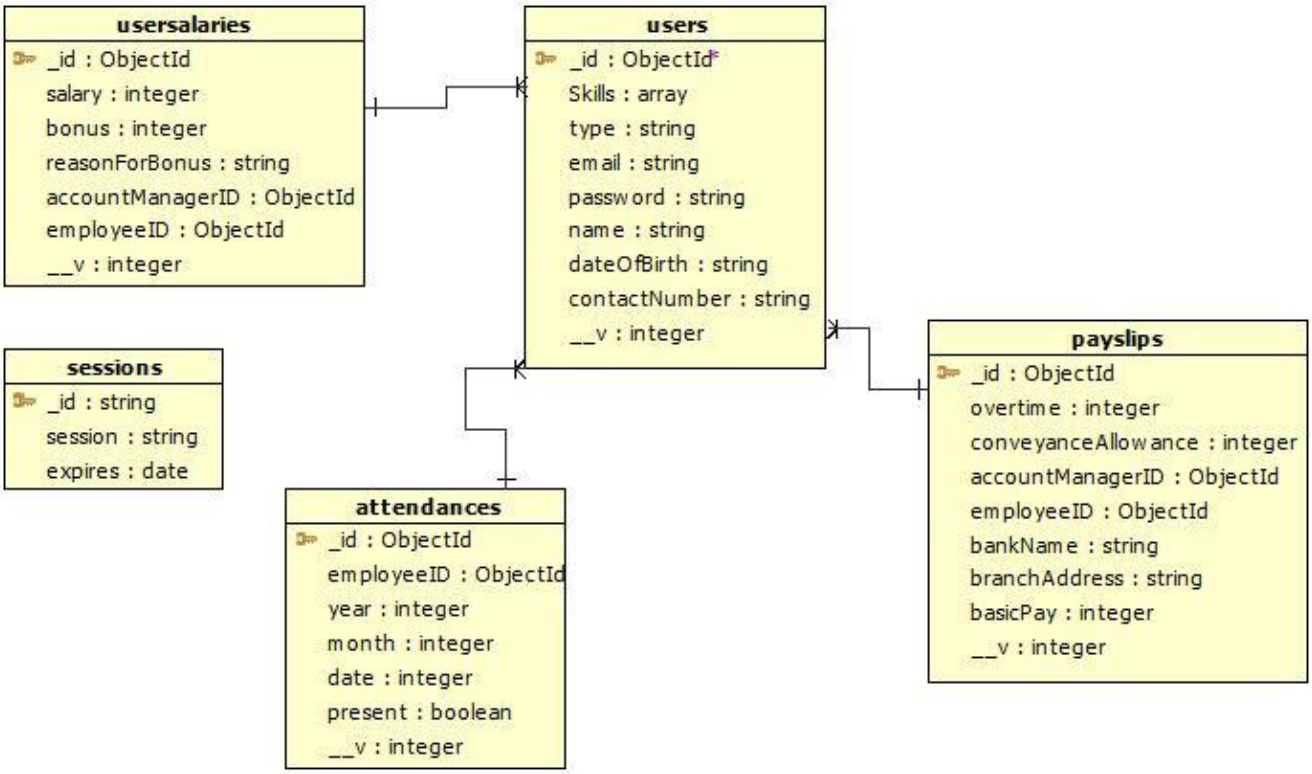


Рисунок 4.7 - Схема бази даних з позначенням псевдо-зв’язків

Для опису особливостей проектування схем в MongoDB, були використані такі поняття:

- Вкладення - повне утримання об’єкта (ів) однієї сутності в іншій.

- Посилання - унікальний ідентифікатор ObjectId зовнішньої сутності.

Використання інструменту Mongoose дозволяє автоматично генерувати ObjectId при вставці вкладеного об'єкта. Наприклад, при додаванні продукту в тип продукту. При проектуванні були визначені деякі особливості проектування схеми БД в MongoDB, які полягають в тому, що:

- При складових псевдо-ключах втрачається гнучкість роботи з даними.

- Зберігання тільки ідентифікатора об’єкта зовнішньої сутності зажадає виконати додатковий запит для отримання всіх даних зовнішньої сутності.

- Вкладення об’єкта сутності має сенс тоді, коли вкладений об'єкт буде зустрічатися разом з батьківським.

- Вкладення об'єкта сутності відображає в реляційній схемі зв’язок один до багатьох.

- Посилання забезпечують більшу гнучкість при частих змінах вкладеної сутності.

- Масив посилань об'єктів сутності відображає в реляційній схемі зв'язок багато до багатьох. Атрибути основної сутності Users, представлені в таблиці 4.3., а на рисунку 4.8 представлено структуру проекту інформаційної системи для зберігання даних.

Таблиця 4.3 - Атрибути основної сутності Users

Найменування атрибуту	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор Users
Skills	array	Практичні навички
type	string	Тип
email	string	email
password	string	Пароль
name	string	Прізвище, ім’я, по батькові
dateOfBirth	string	Дата народження
contactNumber	string	Контактний номер телефону
__v	date	Дата нагадування актуальності
department	string	Підрозділ
designation	string	Посада

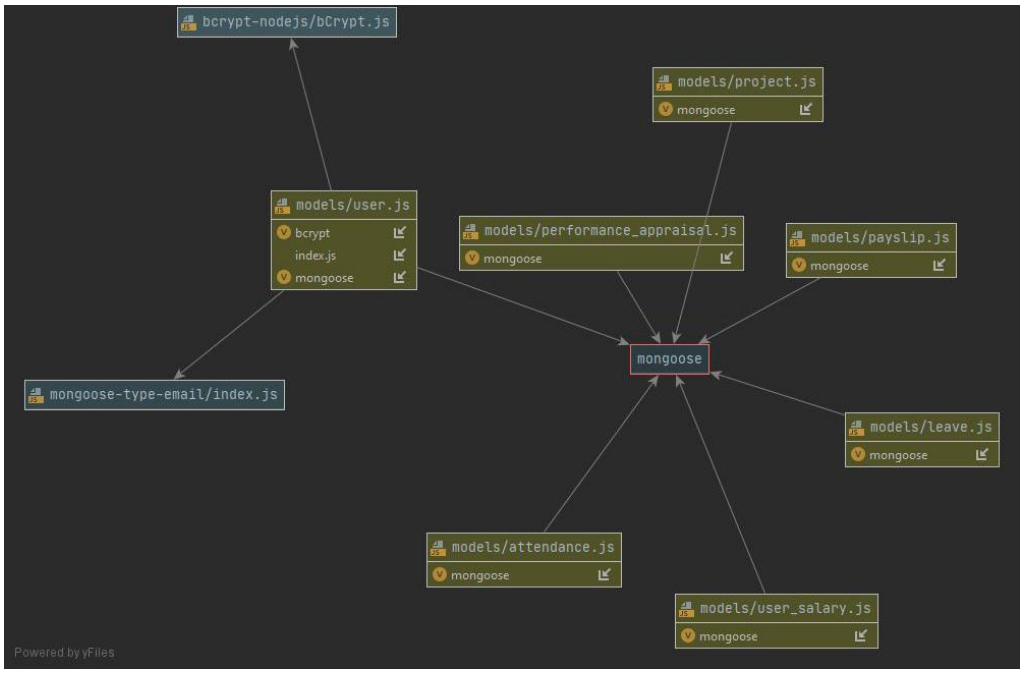


Рисунок 4.8 Структура проекту інформаційної системи для зберігання даних

При запиті користувача відбувається пошук по атрибутах: name, contactNumber, department, designation. Результат повинен повернути інформацію про користувача з відповідними результуючими даними.

4.3 Тестування системи

Розглянемо процедуру тестування системи управління HR-процесами за допопогою Mocha і Chai для його тестування. Mocha - це javascript фреймворк для Node.js (рисунок 4.9), який дозволяє проводити асинхронне тестування. Він створює середовище, в якому ми можемо використовувати свої улюблені assert бібліотеки.

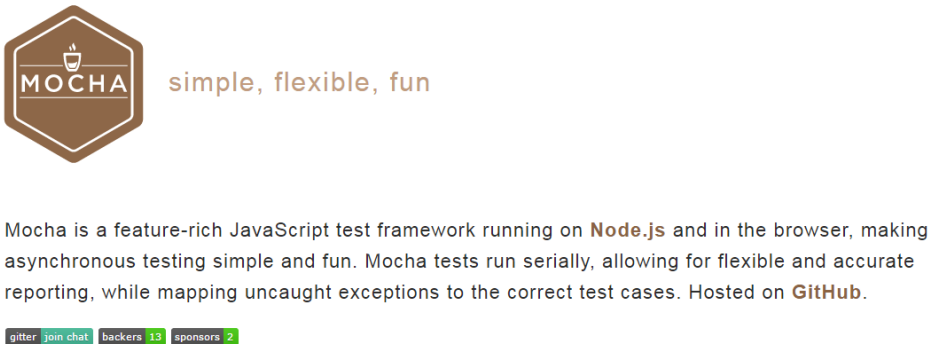


Рисунок 4.9 - Середовище ткстування Mocha

Mocha поставляється з величезною кількістю можливостей. На сайті їх величезний список. Найбільше нам занадобилося наступне:

- проста підтримка асинхронності, включаючи Promise;
- підтримка таймаутів асинхронного виконання before, after, before each, after each (дуже корисно для очищення середовища перед тестами);
- використання будь-якої assertion бібліотеки, яку ви знаходите (в нашому випадку Chai).

Chai: assertion бібліотека (рисунок 4.10). Отже, з Mocha у нас з’явилося середовище для виконання наших тестів. Chai дає нам можливість вибору

інтерфейсу: "should", "expect", "assert". В нашому проекті використано should. У Chai є плагін Chai HTTP, який дозволяє без труднощів тестувати HTTP запити.

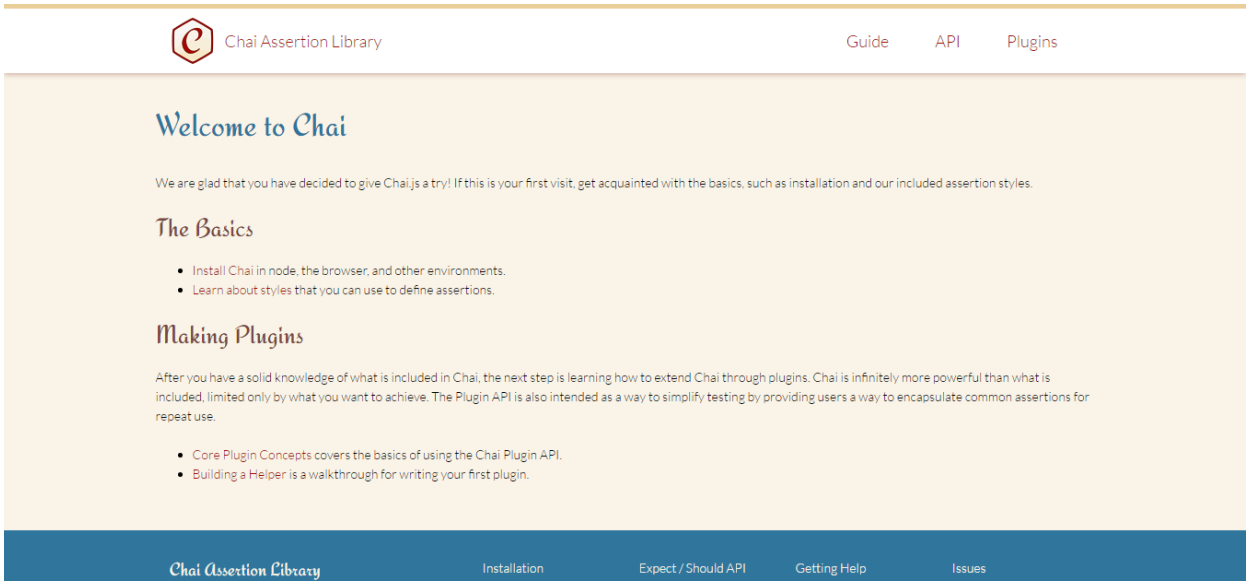


Рисунок 4.10 - Assertion бібліотека Chai

Тестуємо / GET. Chai виконує GET запит і перевіряє, що змінна res задовольняє першому параметру (твердження) блоку it "it should GET all the Documents". А саме, для даного порожнього завдання відповідь повинна бути наступною:

- Статус 200;
- Результат повинен бути масивом.
- Так як база порожня, ми очікуємо що розмір масиву дорівнюватиме 0.

Синтаксис should інтуїтивний і дуже схожий на розмовну мову. В командному рядку виконуємо команду: npm test. Результат виконання команди представлено на рисунку 4.11.

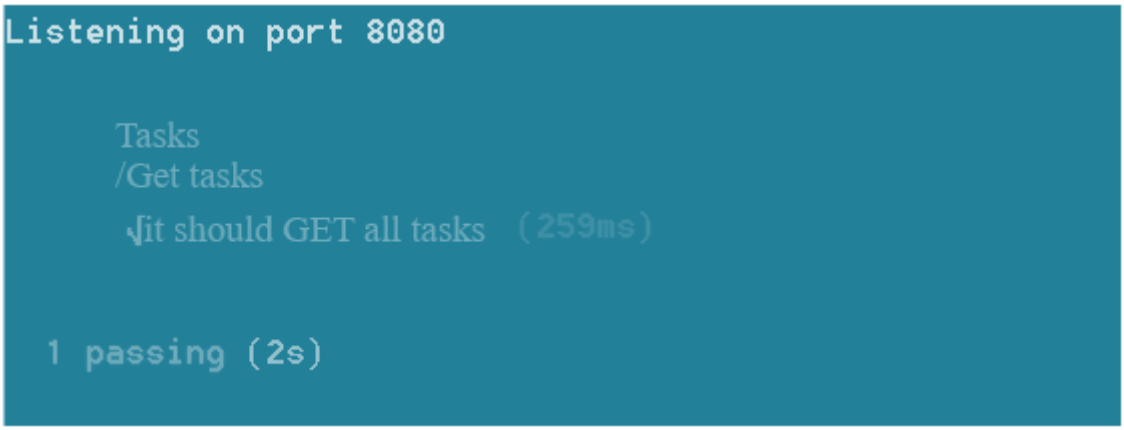


Рисунок 4.12 - Результат виконання команди з використанням Chai

Тест пройшов і результат відображає структуру, яку ми описали за допомогою блоків describe.

Тестуємо / POST. Припустимо ми намагаємося додати реквізиту без поля `describe: сервер не повинен повернути відповідну помилку. Для цього додано код в кінець блоку describe ('Projects'):

```
    describe('/POST Project', () => {
          it('it should not POST a Project without describe field',
(done) => {
              let Project = {
                  title: "Array",
                  Manager: "Крашівський",
                  year: 2020
              }
              chai.request(server)
                  .post('/ Project ')
                  .send(Project)
                  .end((err, res) => {
                      res.should.have.status(200);
                      res.body.should.be.a('object');
                      res.body.should.have.property('errors');
                      res.body.errors.should.have.property('pages');

res.body.errors.pages.should.have.property('kind').eql('required');
                    done();
                  });
       });
```

Тестуємо/ GET /: ідентифікатор. Тепер створюємо реквізит, зберігаємо її в базу та використовуємо id для виконання GET запросу. Добавляємо наступний блок:

```
    describe('/GET/:id Project ', () => {
```

```
          it('it should GET a Project by the given id', (done) => {
            let Document = new Project ({ title: "Max array",
Client: "Крашівський", date: 05.2020, value: 1064 });
            Document.save((err, Project) => {
                chai.request(server)
                .get('/ Project /' + Project.id)
                .send(Document)
                .end((err, res) => {
            res.should.have.status(200);
            res.body.should.be.a('object');
            res.body.should.have.property('title');
            res.body.should.have.property('Manager');
            res.body.should.have.property('date');
            res.body.should.have.property('value');
              res.body.should.have.property('_id').eql(Project.id);
                  done();
                });
            });
          });
      });
```

Через asserts ми переконалися, що сервер повернув всі поля і потрібну реквізиту (id у відповіді від сервера збігається з запитаним) (рисунок 4.13):

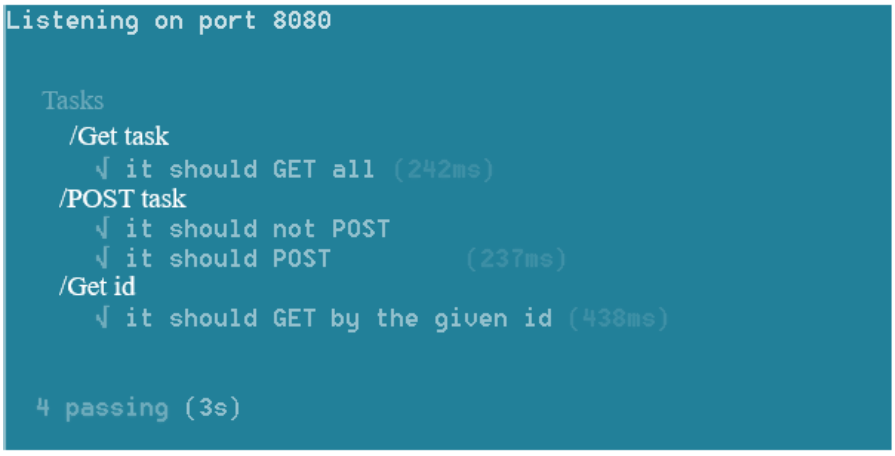


Рисунок 4.13 - Результат виконання команди з використанням Chai

Тестування окремих маршрутів всередині незалежних блоків дозволило отримали хороший вивід, ми написали кілька тестів, які можна повторити за допомогою однієї команди. Для проходження етапу тестування було вирішено розгорнути розроблюваний веб-додаток на хмарному сервісі. Такі сервіси як Heroku і Windows Azure дають можливість безкоштовного використання деяких хмарних ресурсів для своїх потреб. Windows Azure не дає безкоштовного

використання NoSQL бази даних MongoDB, тому розгортання проходило на хмарному сервісі Heroku. Веб-додаток пройшов також наступні види тестування: функціональне тестування; тестування інтерфейсу; юзабіліті тестування.

Функціональне тестування. Функціональне тестування - це тестування програмного забезпечення з метою перевірки можливості бути реалізованим функціональних вимог, тобто здатності програмного забезпечення в певних умовах вирішувати завдання, потрібні користувачам. Функціональні вимоги визначають, що саме робить програмне забезпечення, які завдання воно вирішує. Набір тестів функціонального тестування наведено в таблиці 4.4.

Таблиця 4.4 - Результати функціонального тестування

№	Найменування тесту	Кроки	Очікуваний результат	Результат тестування
1	Реєстрація нового працівника	1. Користувач натискає на посилання «Додати працівників»	Перехід на сторінку введення інформації про працівника.	Успішний
2	Встановлення заробітної плати працівнику	1. Бухгалтер встановлює базову ставку	Відображення змін стосовно оплати праці	Успішний
3	Формування оцінки роботи для працівника	1. Керівник проекту вибирає необхідного працівника 2. Натискає на кнопку «Оцінка роботи»	Відображення оцінки роботи для кожного працівника	Успішний

Тестування інтерфейсу. Розроблений веб-додаток було протестовано з урахуванням сумісності технології WebSocket на браузерах Google Chrome 36.0, Mozilla Firefox 32.0, Opera 20.0, Internet Explorer 10. Всі сторінки відображаються коректно, інтерфейс ідентичний у всіх браузерах.

Юзабіліті-тестування. Юзабіліті-тестування (перевірка ергономічності) - метод оцінки зручності продукту у використанні, заснований на залученні користувачів в якості тестувальників, випробувачів і підсумовуванні отриманих від

них висновків. При проходженні даного тесту були залучені адміністратор і працівники. На занятті менеджереві було запропоновано вирішити такі завдання:

- зареєструвати нового працівника;
- сформувати набір практичних навичок;
- призначити користувача на відповідний проект;

Для працівників були запропоновані наступні завдання:

- підключитися до створеного адміністратором проекту;
- переглянути власний профіль;
- виконати завдання керівника проекта.

Тест пройдено успішно, але з зауваженням: при реєстрації користувача на проект необхідно враховувати практичні навички, якими володіє працівник.

4.4 Інструкція користувача та опис основних функціональних можливостей системи

Після запуску браузера вводимо (якщо не налаштовано заздалегідь системним адміністратором на автоматичне завантаження) адресу нашої системи. У нашому випадку - http://localhost:3000/. У вікні ми бачимо стартову сторінку для авторизації в системі (рисунок 4.14). Після авторизації користувач потрапляє у систему, форма відображення якої залежить від наданих прав доступу, а також від ролі, яка надана користувачеві в процесі його реєстрації в системі. Як зазначалося раніше, у системі передбачено чотири рівні доступу до системи: адміністратор, керівник проекту, працівник, бухгалтер.

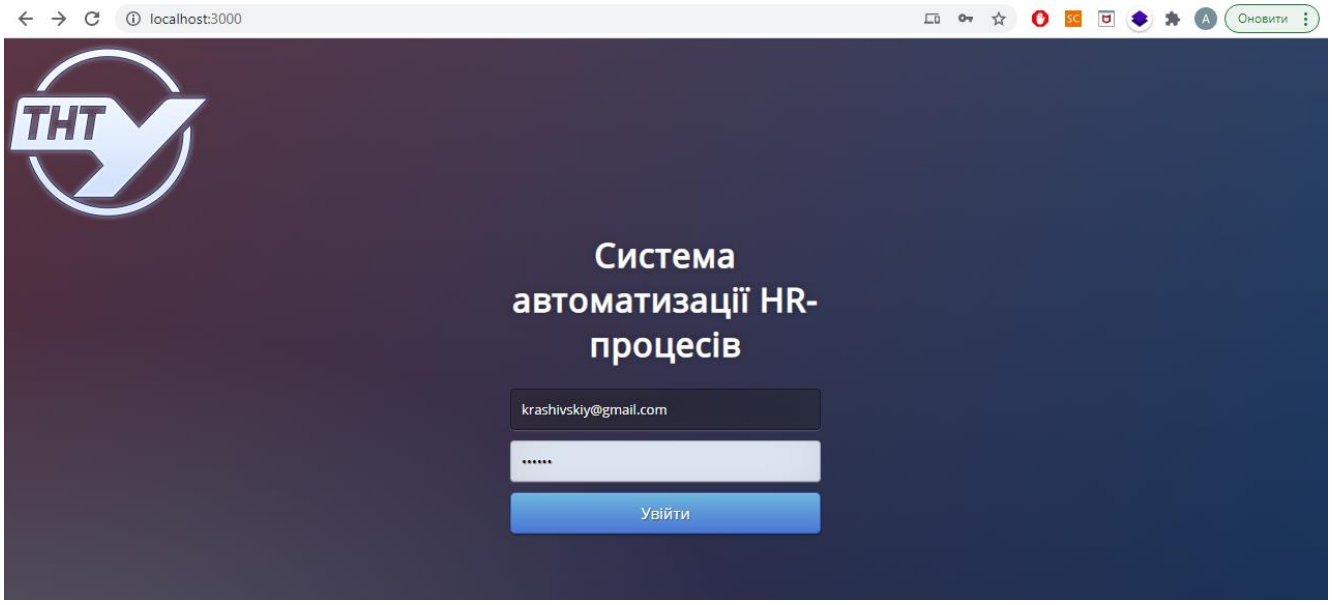


Рисунок 4.14 – Головна сторінка для входу в систему

На рисунку 4.15 представлено головна сторінка системи для адміністратора. У лівій частині сторінки міститься головне меню, за допомогою якого викликаються усі функції системи.

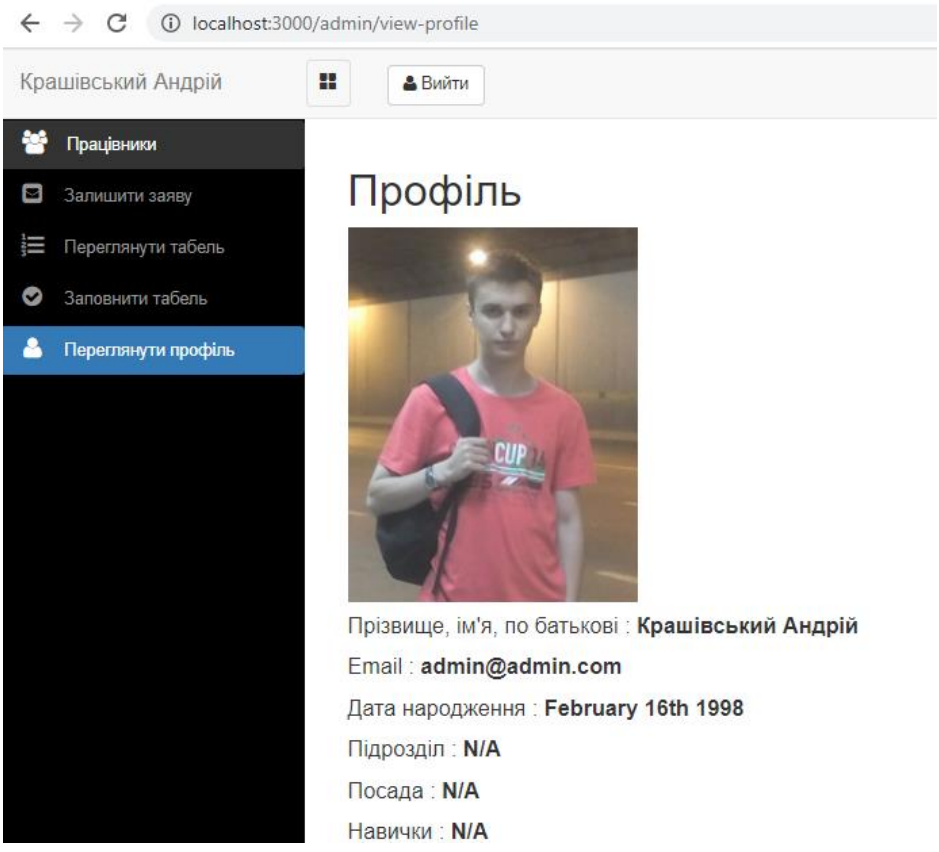


Рисунок 4.22 – Реалізація функціональності для адміністратора

Адміністратор має повний доступ до системи, який включає реєстрацію працівників, прийняти рішення про привілеї для інших працівників, перегляд і зміна обліку відпрацьованого часу, перегляд і зміна заробітної плати працівника, видалення запису або профілю працівника, розподіл і перерозподіл проекту для кожного співробітника, затвердження та відхилення заяви про відпустку працівника.

На рисунку 4.16 представлено форму реєстрації нового працівника в системі. Заповнивши відповідні дані, вся інформація буде збережена у відповідних документах спроектованої MongoDB бази даних.

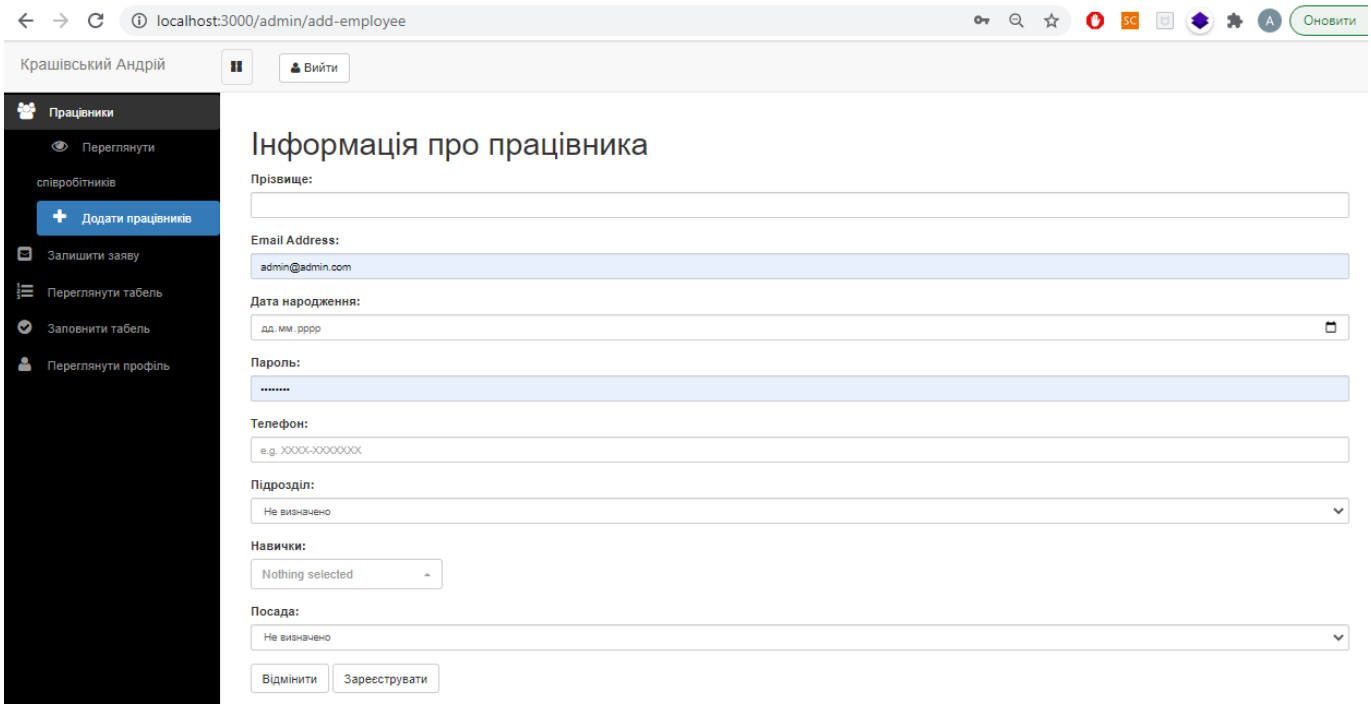


Рисунок 4.16 – Форма реєстрації нового працівника

Обов’язковою умовою успішного додавання працівника є відзначення відповідних практичних навиків, які будуть використовуватися на етапі виконання того чи іншого проекту (рисунок 4.17).

Також адміністратор може переглянути інформацію про всіх працівників, які зараз зареєстровані в системі, їх контактні дані, посаду, підрозділ, проекти на яких вони закріплені.

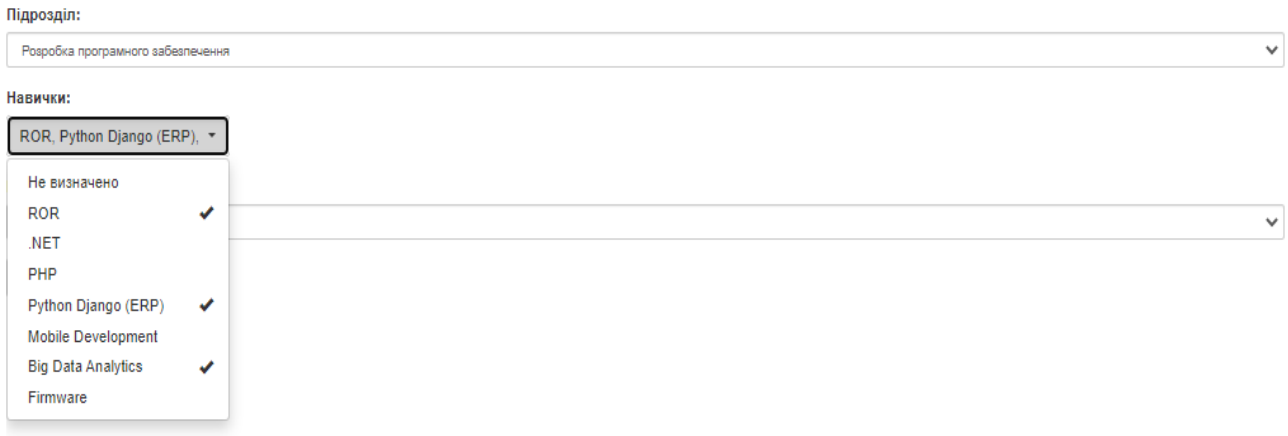


Рисунок 4.17 – Форма відзначення практичних навичок

На рисунку 4.18 представлено сторінку з відображенням інформації про працівників.

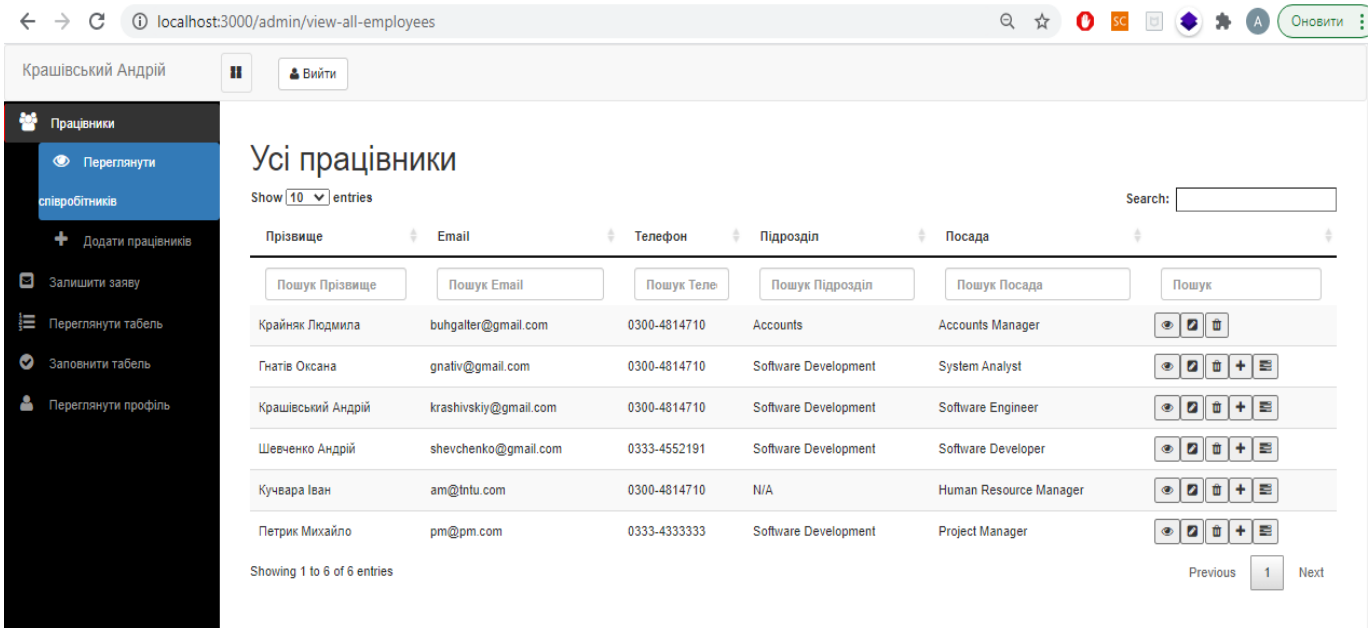


Рисунок 4.18 – Сторінка відображення інформації про працівників, які зареєстровані в системі

Також в системі реалізована можливість перегляду профілю кожного працівника (рисунок 4.19).

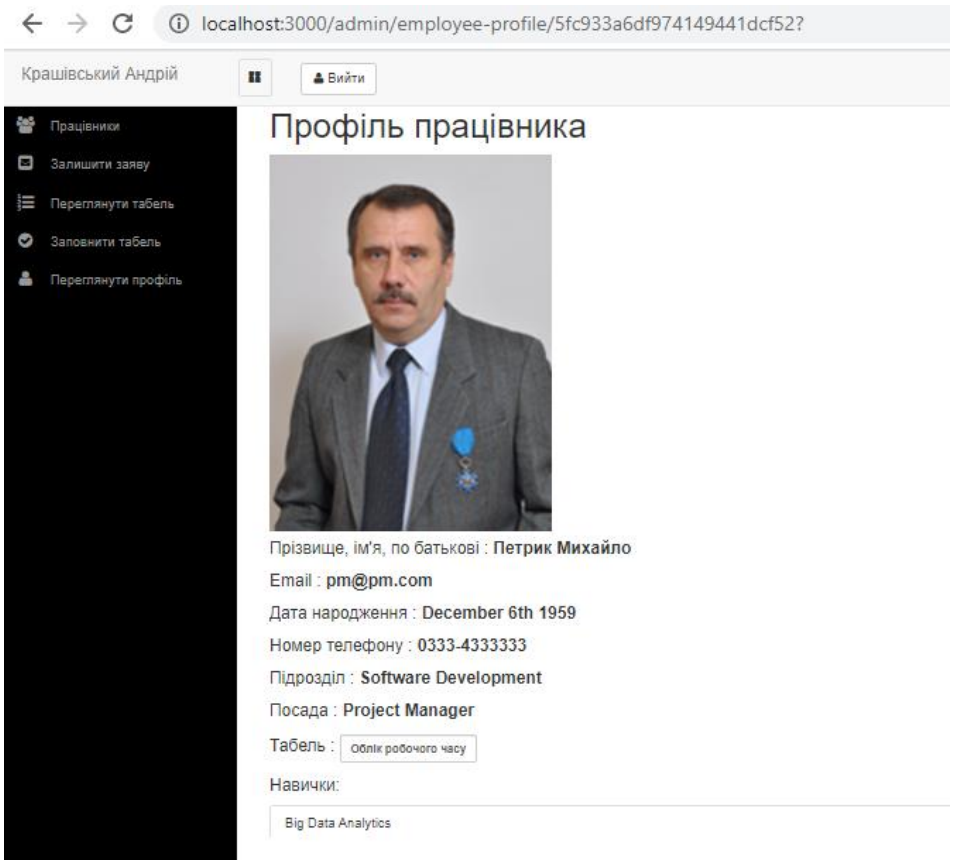


Рисунок 4.19 – Сторінка відображення інформації про працівника

В системі також реалізований функціонал відносно обліку відпрацьованого часу, а також надання відпусток, реєстру відповідних заяв. На рисунку 4.20 представлено форму подачі заяви на відпустку працівником.

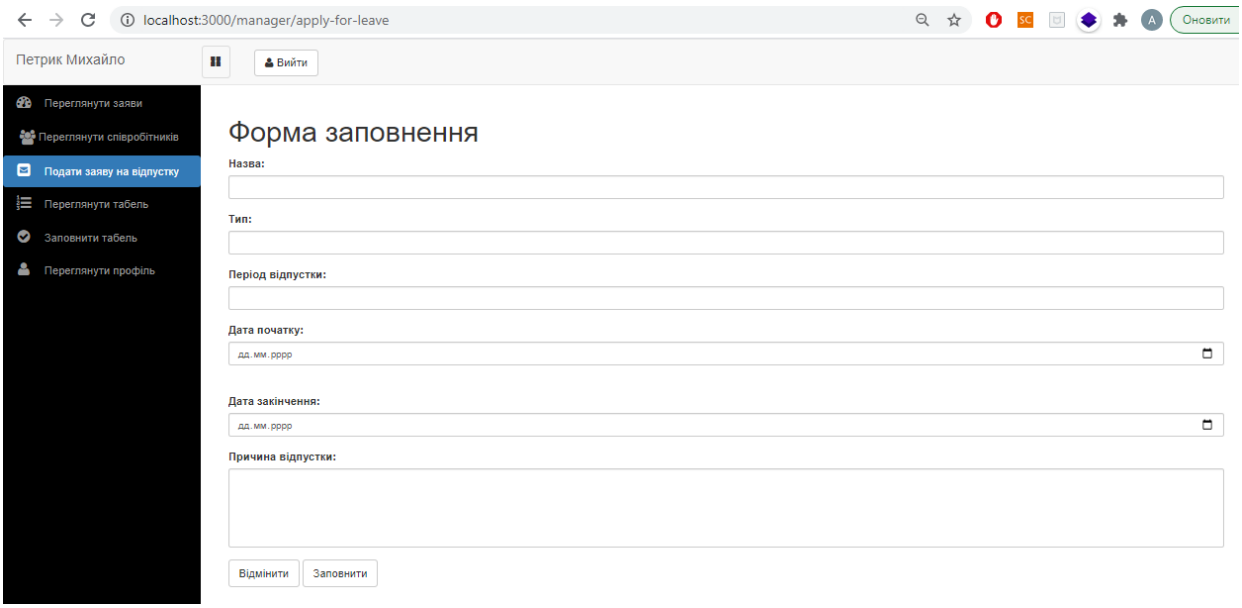


Рисунок 4.20 – Форма подачі заяви на відпустку

Керівник проекту в системі може призначати працівників на відповідні проекти, а також здійснювати оцінку їх діяльності, формуючи відповідну систему якісної оцінки кожного працівника. Відповідну форму представлено на рисунку 4.21.

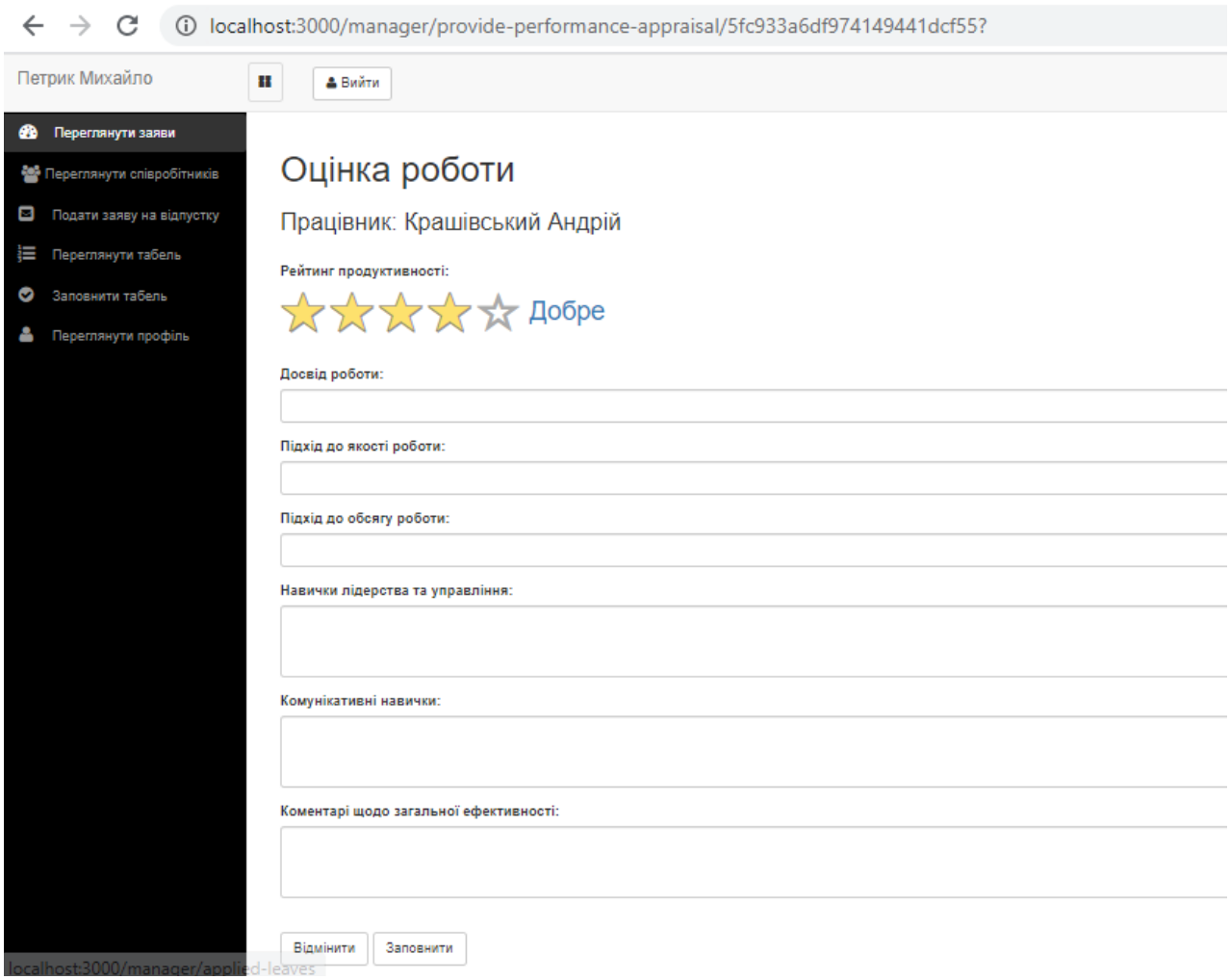


Рисунок 4.21 – Форма оцінки результативності роботи працівників

Особлива увага в системі приділена питанням обліку відпрацьованого часу та оплати праці. Кожен працівник зможе заповнити свій табель обліку робочого часу, переглянути його історію, переглянути поточну заробітну плату, переглянути поточний профіль працівника (включаючи освітню та виробничу історію), переглянути всі проекти в межах організації, переглянути інших співробітників, які ділять з ним один проект, подати заявку на відпустку та переглянути статус відповідної заяви на відпустку.

На рисунку 4.22 представлено сторінку з інформацією про відпрацьований час працівника по днях.

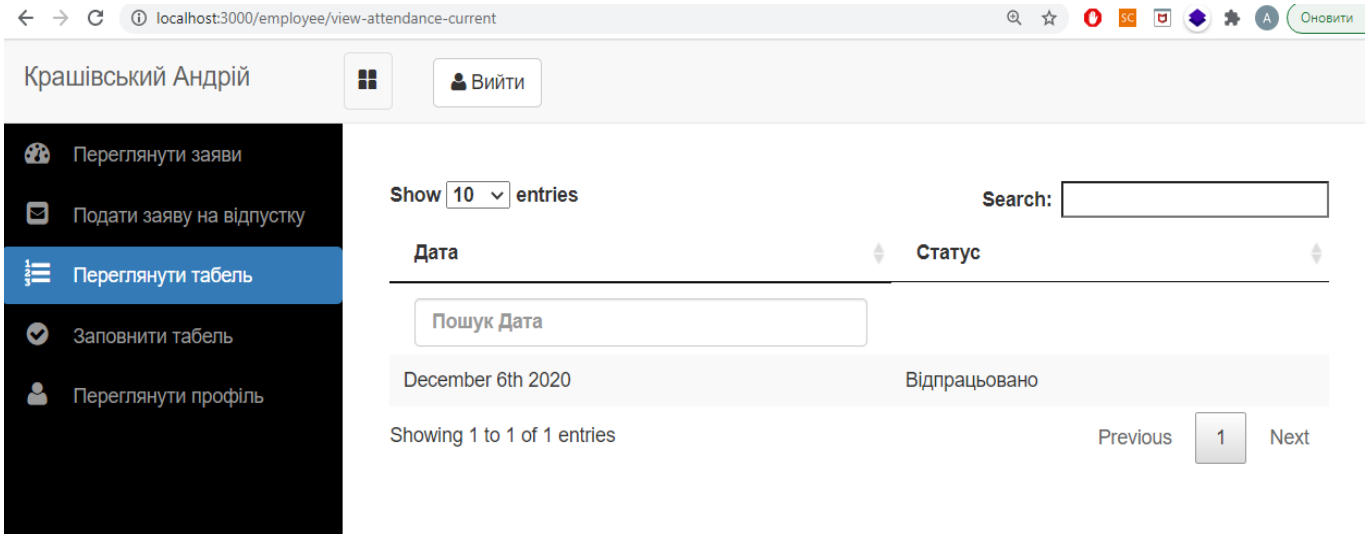


Рисунок 4.22 – Сторінка перегляду відпрацьованого часу працівником

Відповідно до розмежування прав доступу до системи, то усі аспекти з питань обліку та оплати праці працівникам належить користувачам, які входять в бухгалтерську групу (рисунок 4.23).

Працівник бухгалтерії може сформувати відомість про оплату праці для кожного працівника відповідно до обраного банку, встановити премію для працівника, встановити заробітну плату працівника (рисунок 4.24), збільшити заробітну плату працівника (рисунок 4.25), надіслати відомість про оплату по електронній пошті кожному працівникові (рисунок 4.26). Розглянемо детальніше ці бізнес-процеси.

На рисунку 4.23 представлено сторінку з інформацією про оплату праці працівників. Ця сторінка доступна тільки працівникам бухгалтерії.

Працівник бухгалтерії відповідно до чітко визначених критеріїв може вставновити зарплату для працівника, яка буде включати базову ставку, а також розмір премії. Відповідну інформацію представлено на рисунку 4.23.

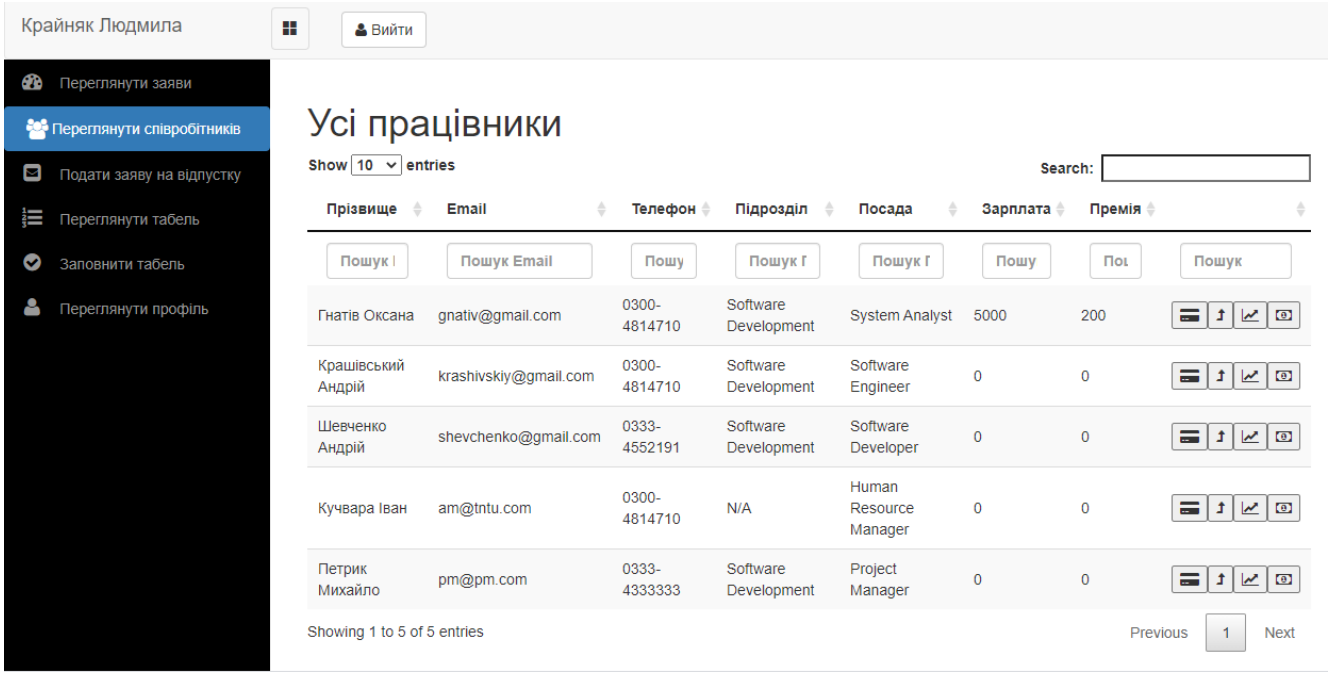


Рисунок 4.23 – Сторінка обліку оплати праці співробітників

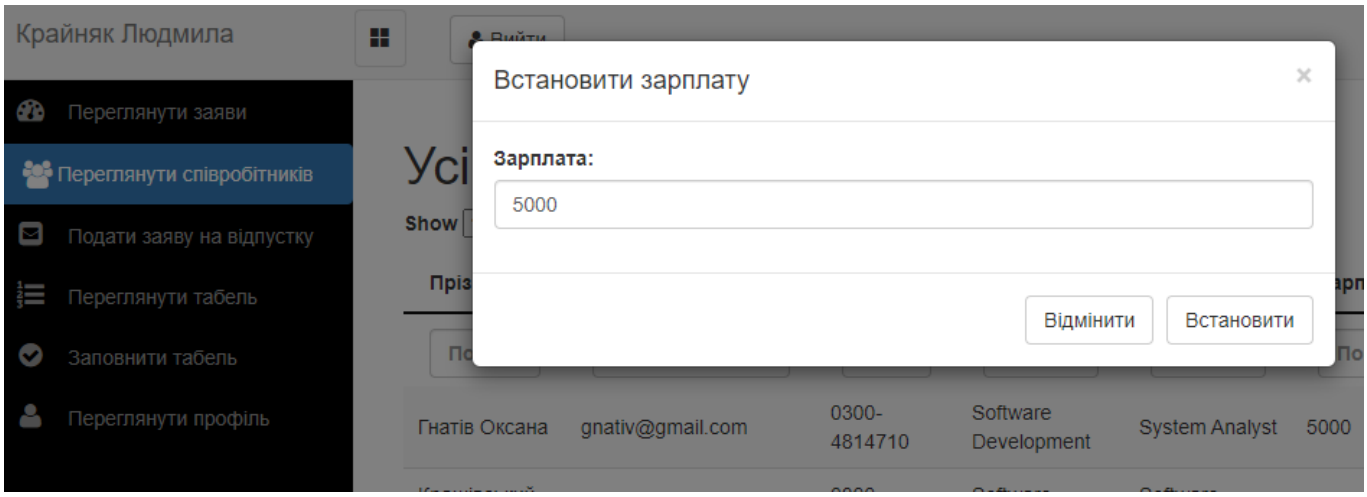


Рисунок 4.24 – Сторінка встановлення зарплати для окремого працівника

В системі бередбачена можливість формування бонусів для працівників за рахунок відповідних премії. Відповідно до поданих пропозицій керівниками проектів ці премії мають диференційований характер. На рисунку 4.25 представлено форму встановлення премії для працівника.

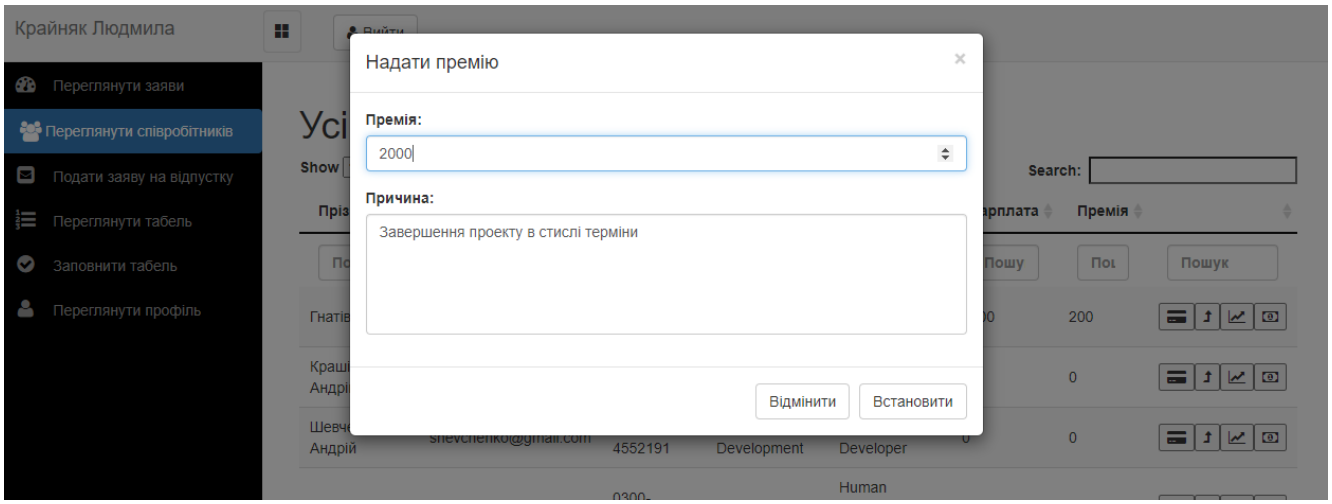


Рисунок 4.25 – Сторінка встановлення премії для окремого працівника

На рисунку 4.26 представлено сторінку формування платіжної відомості, яка включає вибір банку отримувача, адресу, інформацію про відпрацьований час окремим працівником, а також розмір нарахованої заробітної плати.

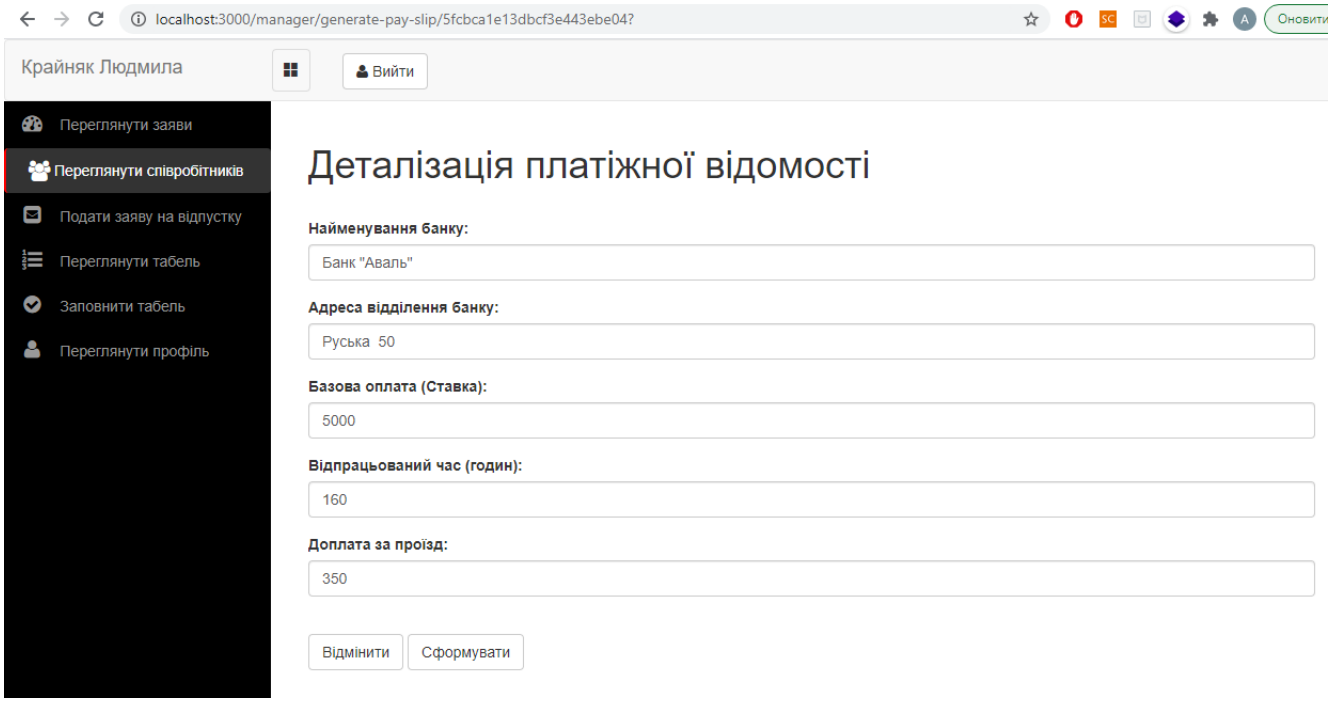


Рисунок 4.26 – Сторінка формування платіжної відомості для окремого працівника

Виходячи з описаного вище, можна підсумувати, що розроблена інформаційна система дозволяє:

- реалізацію окремих облікових записів користувачів для всіх адміністраторів та працівників;

- різні представлення про дані та чіткий доступ до даних для учасників на основі їхніх привілеїв;

- реєстрація працівника / адміністратора;

- управління відпрацьованим часом всіх працівників та їх заробітною платою.

- ведення обліку освітнього та виробничого досвіду працівника;

- управління поточними розподіленими проектами працівників у межах організації.

4.5 Експериментальні дослідження моделей управління реплікаціями на етапі проектування системи автоматизації HR-процесів

Розроблені моделі були використані при проектуванні інформаційної системи управління HR-процесами на базі Node.js, яка повинна забезпечувати зберігання і обробку великого масиву даних про працівників, етапи виконання проектів, облік відпрацьованого часу та нарахування з оплати праці.

Завданням дослідження був вибір параметрів реплікації (N, W, R) розподіленої бази даних MongoDB з урахуванням специфіки предметної області та вимог замовника до показників продуктивності, узгодженості та відмовостійкості. У процесі вирішення поставленого завдання були реалізовані наступні етапи:

- опис предметної області;

- підстави для відмови від реляційної моделі даних на користь технології NoSQL;

- визначення доступних варіантів MongoDB, які можуть бути використані для реалізації досліджуваної інформаційної системи;

- побудова структури сховища інформаційної системи (опис агрегатів);

- оцінка показників продуктивності, узгодженості та відмовостійкості інформаційної системи на етапі її проектування з використанням моделей, розроблених в попередніх розділах роботи;

- вибір параметрів реплікації на підставі вимог, що пред’являються до продуктивності, узгодженості та відмовостійкості ІС, з урахуванням виконаних оцінок.

Проектована система призначена для збору, передачі, зберігання і аналізу інформації про облік відпрацьованого часу, оплати праці, а також пов'язаних з ними даних по оцінці якості виконання проектів. Вона дозволяє обробляти інформацію по проектах, які виконують працівники (модуль управління проектами), модуль обліку відпрацьованого часу і пов’язаних з ними нарахувань з оплати праці (модуль організації роботи з оплати праці). Однією з відмінних рис системи є інтеграція різних компонентів між собою, а також інтеграція кожного з цих компонентів з проектними даними.

Модуль управління проектами призначений для збору інформації як по індивідуальних проектах (зазвичай це особливо важливі компоненти, що вимагають негайного повідомлення, такі як впровадження проекту або реінженерія), так і по командних проектах (зазвичай це завдання, що вимагають колективних повідомлень, наприклад, у вигляді групових мітінгів).

Інформація про індивідуальний внесок в командне виконання проекту включає дані про практичні навички, дані про управління проектом, завдання (попередні, поточні, кінцеві), інтерпретація цих даних.

Модуль з обліку робочого часу містить інформацію про відпрацьовані години, що затрачені на виконання того чи іншого проекту.

Модуль з оплати праці дозволяє співробітникам отримати доступ до інформації про розмір відповідних нарахувань. В свою чергу модулі дозволяють отримати доступ до упраління проектами, основних етапів їх виконання та відповідних витрат. Таким чином, забезпечується інтеграція модулів між собою. Специфіка предметної області вимагає сукупного аналізу множини чинників для проведення якісного упраління проектами. Вже згадана система складається з двох блоків обробки даних, представлених на рисунку 4.27.

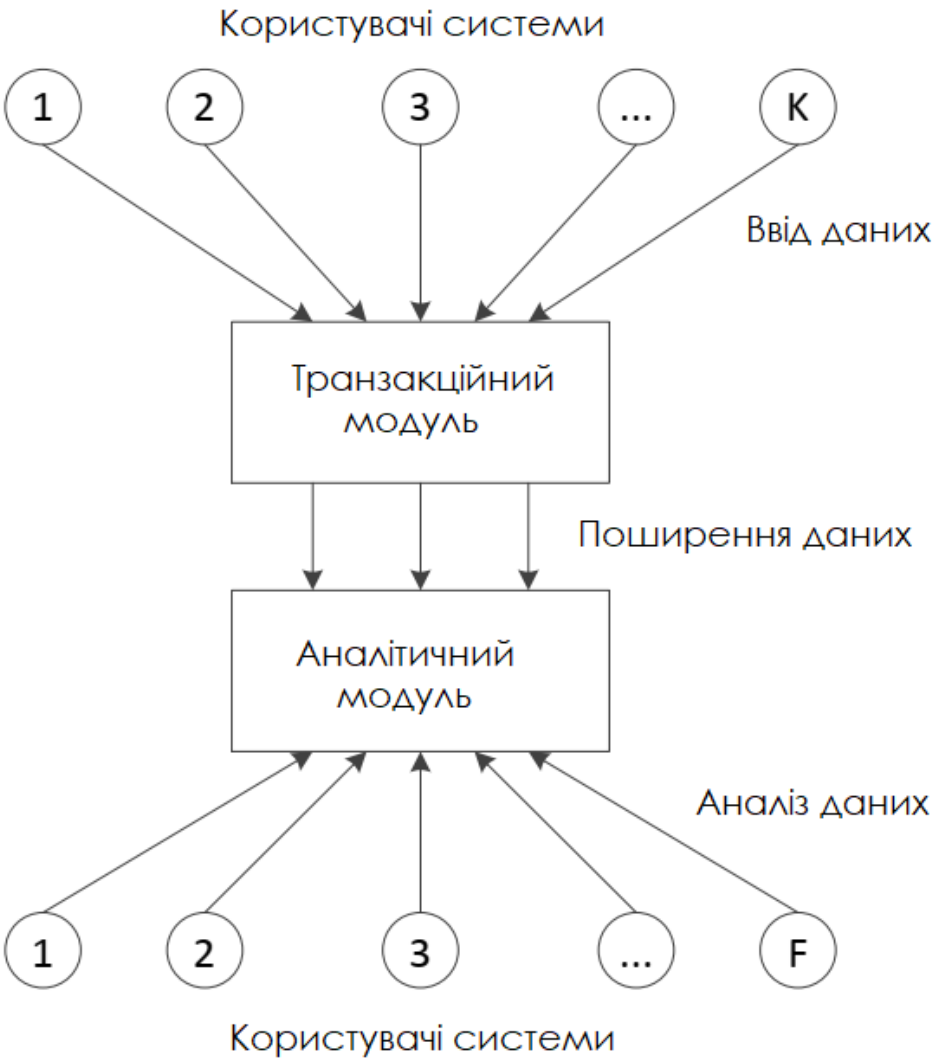


Рисунок 4.27 – Структура модулів системи управління HR-процесами на етапі проектування

1. Транзакційний модуль - вузол, який займається збором, обробкою і подальшою передачею даних на аналітичний модуль. Даний елемент системи може працювати на основі будь-якої надійної реляційної СУБД, що підтримує ACID транзакції.

2. Аналітичний модуль - набір (кластер) вузлів, що займаються аналітичною обробкою даних. Це один з основних компонентів системи. Він дозволяє представляти зібрані дані у вигляді різного роду звітів, будувати різноманітні діаграми і відображати інформацію в прив’язці до географічних координат на карті. Даний елемент системи дозволяє паралельно обробляти аналітичні запити і видавати звіти різного рівня складності.

У штатному режимі роботи системи число запитів, що надходять до аналітичного модулю невисоке, з навантаженням цілком може впорається один вузол. Однак під час великого корпоративного проекту навантаження на систему може значно зрости.

До недавнього часу доступ до бази даних найчастіше здійснювався з використанням класичної реляційної системи управління базами даних. В даний час левова частка інформації зберігається в СУБД. Це пояснюється зовнішньою простотою реляційної моделі, наявністю такого потужного і в той же час простого для вивчення мови доступу до даних, як SQL, а також наявністю оптимізаторів виконання транзакцій і запитів до бази даних, які забезпечують їх «дроблення» на більш дрібні завдання і паралельну реалізацію цих робіт на багатопроцесорних або багатомашинних комплексах.

Розглянемо основні проблеми організації даних, які можуть виникнути при використанні реляційної бази даних в процесі реалізації аналітичного модуля:

1. Проблема втрати відповідності та низької продуктивності. Як зазначалося в розділі 1, вона проявляється в тому, що реляційні бази даних не дозволяють зберігати агрегати. Наприклад, щоб відобразити інформацію про керівника проекту, місце перебування, практичні навички і т.д., програміст повинен зібрати в оперативній пам’яті дані з багатьох таблиць. При значному зростанні числа таблиць розробник часто просто забуває призначення тієї чи іншої таблиці, ускладнюється зв'язування таблиць при виконанні запиту, тобто істотно ускладнюється формування агрегату. Операції з'єднання багатьох таблиць виконуються порівняно повільно. Теоретично можливий варіант використання однієї плоскої таблиці. Але в такому випадку таблиці не буде знаходитися в третій нормальній формі і буде мати недоліки, такими як надмірність, потенційна суперечливість і т.д. При цьому обсяг бази даних може зрости на порядок.

2. Проблема сегментації даних. Як зазначалося в розділі 1, зі збільшенням обсягу збережених даних виникає завдання фрагментації таблиць бази даних по різних серверах, об’єднаних в кластер. Але паралельні системи реляційних баз даних демонструють невисоку масштабованість [5]. При виконанні складних запитів

продуктивність системи істотно зменшується в результаті міжмашинного обміну даними між серверами кластера [6].

3. Проблема забезпечення відмовостійкості. Як зазначалося в розділі 1, в паралельних системах баз даних число реплік невелике, що не може гарантувати високу відмовостійкість. Більш того, при відмові будь-якого вузла необхідно перезапускати всю систему. Бази даних NoSQL не мають зазначених недоліків, тому в даному проекті доцільно було використовувати базу даних цього типу.

В рамках даного магістерського дослідження оцінювалися наступні показники функціонування аналітичного модуля:

1) середній час очікування вимоги на читання відновлення W реплік для режиму суворого узгодження реплік;

2) ймовірність того, що клієнт прочитає застарілий запис за час розповсюдження оновлень записів по її N-W реплік для режиму узгодження реплік в кінцевому рахунку;

3) ймовірність відмови в доступі до запису БД.

Були визначені вимоги до режимів узгодження реплік аналітичного модуля системи «Управління HR-процесами»: сегмент з обліку працівників - узгодженість в кінцевому рахунку, сегмент з оплати праці - сувора узгодженість.

Замовник встановив розмір кластера в 15 вузлів. Додавання / видалення вузлів (серверів) виконується однією командою, при цьому дані перерозподіляються між усіма вузлами автоматично у фоновому режимі. Це істотно спрощує процес зміни розміру системи. Віртуальні вузли мають однакову продуктивність. Згідно [23] рекомендується налаштовувати число віртуальних вузлів в кільці так, щоб на кожному вузлі зберігалося мінімум 10 розділів (віртуальних вузлів). Тому встановимо розмір кільця, рівним 256. Для аналізу були використані наступні значення характеристик апаратних ресурсів:

- інтенсивність читання даних з оперативної пам'яті - 8000 КБ/с;

- продуктивність процесора - 2000 млн операцій за с.

В період формування звітів з оплати праці мережу передачі даних і дискові масиви можуть бути сильно перевантажені, тому були використані наступні параметри:

- інтенсивність передачі даних усередині мережі - 1 Гбіт/с (Стандартна швидкість);

- завантаження мережі передачі даних - 0.6;

- інтенсивність дискового введення / виводу - 40 Мбайт/с;

- кількість ремонтних бригад кластера - 1;

- час напрацювання на відмову одного узла- 90 днів;

- при відмові вузла: ймовірність випадкового збою операційної системи 0.835, ймовірність збою апаратного забезпечення 0.15, ймовірність відмови дисків - 0.015;

- час перезапуску операційної системи - 10 хв;

- час відновлення апаратного забезпечення вузла - 4 год;

- час відновлення дисків - 5 год;

Розглянемо проблеми узгодженості, які можуть виникнути при роботі з системою:

1. Може виникати затримка читання записів з сегмента «Облік відпрацьованого часу», тому що в період здачі проектів інтенсивність читання і поновлення даних сегмента висока.

2. При високій інтенсивності читання даних з сегменту «Управління проектами» користувачам може бути повернута неактуальна інформація (застарілі записи), тому що даний сегмент узгоджений в кінцевому рахунку.

Дослідження показників узгодженості та відмовостійкості було виконано за допомогою розробленого програмного забезпечення. На рисунку 4.28 показані залежності середнього часу очікування (T) вимоги на читання закінчення поновлення W реплік в залежності від інтенсивності надходження вимог на читання записів з відповідного сегмента до однієї репліки (λ). $W = R = N / 2 + 1$, якщо N парне, і $W = R = (N + 1) / 2$, якщо N непарне.

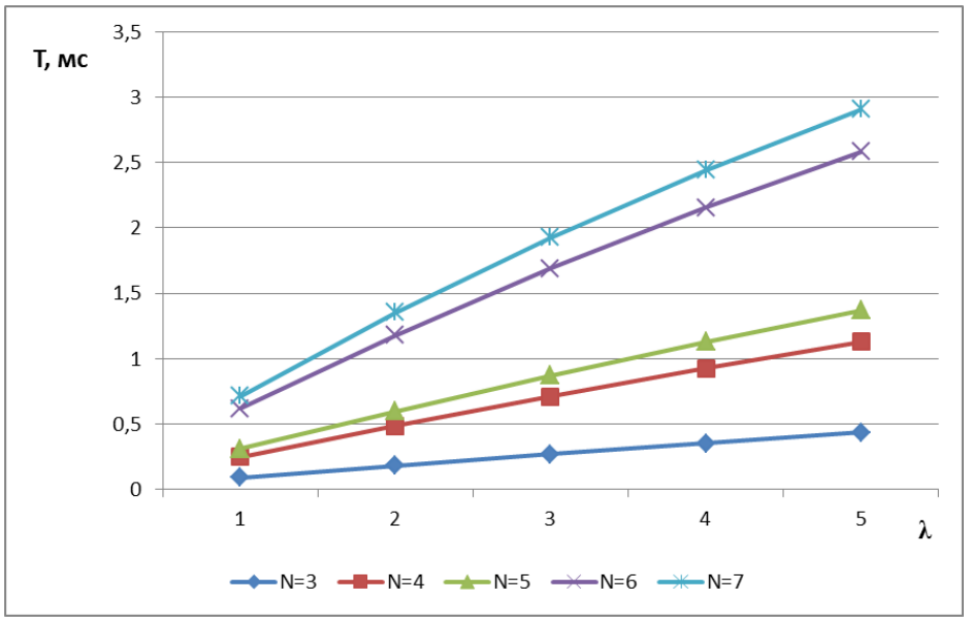
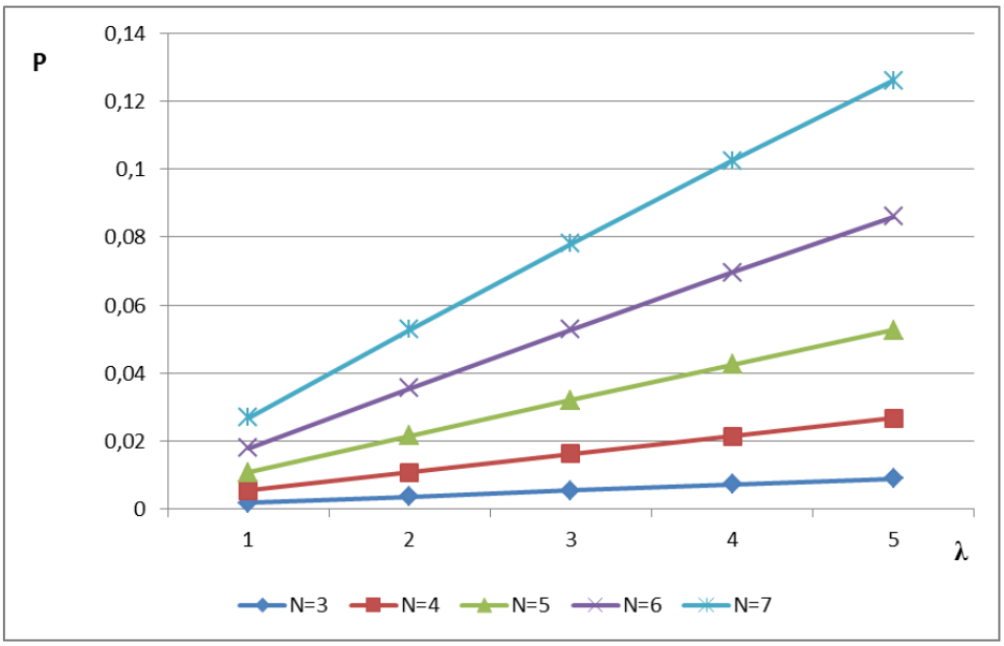


Рисунок 4.29 - Залежності середнього часу очікування T (мс) від інтенсивності надходження вимог на читання λ (1/с) при різних значеннях числа реплік запису N

На рисунку 4.30 наведені залежності ймовірності *P* того, що клієнт прочитає застарілий запис за час розповсюдження оновлень записів по її *N*-*W* реплік сегмента (*W* = *R* = *1*).

Вибір параметрів реплікації сегментів сховища MongoDB був виконаний на підставі вимог, що пред'являються до затримки читання, узгодженості та відмовостійкості. Замовник системи встановив наступні вимоги:

1. Затримка читання (час очікування вимоги на читання закінчення поновлення *W* реплік) - не більше 2 мс (це важливо, якщо при виконанні запиту читається велике число записів).



Рисуно 4.30 - Залежність ймовірності *P* від інтенсивності надходження вимог на читання до одного вузла кластера *λ (1 / c)* при різних значеннях числа реплік запису *N*

2. Ймовірність доступу до неузгоджених даних (ймовірність того, що клієнт прочитає застарілий запис за час поширення оновлень записів по її N-W реплік) - не більше 0.1.

В результаті проведених експериментальних досліджень отримано залежності середнього часу очікування читання записів з БД, ймовірності читання застарілих записів, ймовірності відмови в доступі до запису БД від інтенсивності надходження вимог на читання.

Таким чином розроблена інформаційна система управління HR-процесами на базі Node.js наочно показує сумісність інформаційних технологій та демонструє працюючий програмний продукт, що відповідає сучасним вимогам подібного роду додатків і є конкурентоспроможними на ринку інформаційних технологій.

5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДВИЧАЙНИХ СИТУАЦІЯХ

5.1 Охорона праці

Охорона праці – це система правил і заходів, які забезпечують безпечну роботу на даному виробництві. При роботі на металорізальному верстаті необхідно передбачити ряд вимог, які б дозволили працюючому виконувати поставлене перед ним завдання в умовах, які передбаченні конструкторськими документами .

Функціями охорони праці є дослідження санітарії та гігієни праці, проведення заходів щодо зниження впливу шкідливих факторів на організм працівників у процесі праці. Основним методом охорони праці є використання техніки безпеки. При цьому вирішуються два основні завдання: створення машин і інструментів, при роботі з якими виключена небезпека для людини, і розробка спеціальних засобів захисту, що забезпечують безпеку людини в процесі праці, а також проводиться навчання працюючих безпечним прийомам праці та використання засобів захисту, створюються умови для безпечної роботи .

Цілі і завдання охорони праці - звести до мінімуму ймовірність нещасних випадків та професійних захворювань працюючих з одночасним забезпеченням нормальних умов праці при її максимальній продуктивності.

Небезпечний виробничий фактор - такий фактор виробничого процесу, вплив якого на працюючого приводить до травми або різкого погіршення здоров'я .

Шкідливий виробничий фактор - це фактор трудового процесу або умов навколишнього середовища, який може надати шкідливий вплив на здоров'я і працездатність людини. Тривалий вплив на людину шкідливого виробничого фактора призводить до захворювання .

Небезпечні та шкідливі виробничі фактори підрозділяються по природі дії на наступні групи: фізичні, хімічні, біологічні, психофізіологічні.

Основні небезпечні і шкідливі виробничі фактори, що можуть впливати на оператора (користувача):

а) фізичні:

- підвищений рівень електромагнітного випромінювання;

- підвищений рівень рентгенівського випромінювання;

- підвищений рівень ультрафіолетового випромінювання;

- підвищений рівень інфрачервоного випромінювання;

- підвищений рівень статичної електрики;

- підвищений рівень запиленості повітря робочої зони;

- знижена чи підвищена вологість повітря робочої зони;

- знижена чи підвищена рухомість повітря робочої зони;

- підвищений рівень шуму на робочому місці (від вентиляторів, процесорів, аудіоплат, принтерів);

- підвищений чи знижений рівень освітленості;

- підвищений рівень засліпленості;

- нерівномірність розподілу яскравості в поле зору;

- підвищена яскравість світлового зображення;

- підвищений рівень пульсації світлового потоку;

- ураження електричним струмом;

б) хімічні:

- підвищений вміст у повітрі робочої зони двоокису вуглецю, озону, аміаку, фенолу, формальдегіду;

в) психофізіологічні:

- напруга зору;

- напруга уваги;

- інтелектуальні навантаження;

- емоційні навантаження;

- тривалі статичні навантаження;

- монотонність праці;

- великий обсяг інформації, оброблюваної в одиницю часу;

- нераціональна організація робочого місця;

г) біологічні:

- підвищений вміст у повітрі робочої зони мікроорганізмів.

Відповідно до стандартів, під умовами праці розуміється – це сукупність чинників виробничого середовища трудового процесу, що впливають на здоров'я та працездатність людини в процесі предметної діяльності.

Чинники, що формують умови праці, можна розділити на наступні групи: санітарно-гігієнічні; психофізіологічні; естетичні; соціально-психологічні; організаційно-економічні.

Перелічені групи чинників умов праці складають основу виробничої обстановки.

Стіни приміщень для роботи з ПК мають бути пофарбовані чи обклеєні шпалерами пастельних кольорів з коефіцієнтом відбиття 40 - 60 %. У випадках, коли такі приміщення зорієнтовані на південь, вікна повинні обладнуватися сонцезахисними пристроями (жалюзі, штори і т. п.).

Для освітлення приміщень з ПК необхідно використовувати люмінесцентні світильники. Освітленість робочих місць у горизонтальній площині на висоті 0,8 м від підлоги повинна бути не менше 400 лк. Вертикальна освітленість у площині екрану не більше 300 лк.

У приміщеннях для роботи з ПК необхідно проводити щоденне вологе прибирання та регулярне провітрювання протягом робочого дня. Видалення пилу з екрану необхідно проводити не рідше одного разу на день.

Робочі місця для працюючих з дисплеями необхідно розташовувати таким чином, щоб до поля зору працюючого не потрапляли вікна та освітлювальні прилади. Відео термінали повинні встановлюватися під кутом 90 - 105 градусів до вікон та на відстані, не меншій 2,5 - 3 м від стіни з вікнами.

До поля зору працюючого з дисплеєм не повинні потрапляти поверхні, які мають властивість віддзеркалювання. Покриття столів повинне бути матовим з коефіцієнтом 0,25 - 0,4.

Відстань між робочими місцями з ПК повинна бути не меншою 1,5 м у ряду та не меншою 1 м між рядами. ПК повинні розміщуватися не ближче 1 м від джерела тепла.

Відстань від очей користувача до екрану повинна становити 500 - 700 мм, кут зору - 10 - 20 градусів, але не більше 40 градусів, кут між верхнім краєм відео терміналу та рівнем очей користувача повинен бути меншим 10 градусів. Найбільш вигідне є розташування екрану перпендикулярно до лінії зору користувача.

З метою уникнення перевантаження організму робочий день користувача ПК повинен проходити у раціональному режимі праці та відпочинку, який передбачає дотримання регламентованих перерв, їх активне проведення, систематичне проведення виробничої гімнастики, рівномірний розподіл завдань.

Загальний час роботи з відео терміналом не повинен перевищувати 50% тривалості робочого дня. Якщо виконання роботи пов'язане тільки з використанням комп'ютера, то при неможливості зміни діяльності необхідно робити перерви та паузи.

Для робіт, які виконуються з великим навантаженням, слід робити 10 - 15 хвилинну перерву через кожну годину, для мало інтенсивної роботи такі перерви слід робити через 2 години.

Форми та зміст перерв можуть бути різними: виконання альтернативних допоміжних робіт, які не вимагають великого напруження, приймання їжі та ін. На початку перерв виконується гімнастика для очей, під час однієї з перерв рекомендується проведення загальної гімнастики.

Виконання фізичних вправ з нормативним навантаженням протягом робочого дня рекомендується індивідуально, залежно від відчуття втоми. Гімнастика повинна бути спрямована на корекцію вимушеної пози, покращення кровообігу, часткову компенсацію дефіциту рухливої активності.

5.2 Безпека в надзвичайних ситуаціях

Оператор ЕОМ зобов’язаний:

у всіх випадках виявлення пошкодження проводів електричного живлення, несправності заземлення та інших пошкодженнях електрообладнання, виникненні запаху гарі, диму – негайно вимкнути електричне живлення і повідомити про аварійну ситуацію свого безпосереднього керівника й чергового електрика;

при попаданні людини під електричну напругу негайно звільнити її від дії струму шляхом вимкнення електричного живлення, до прибуття лікаря надати потерпілому долікарську медичну допомогу;

при будь-яких випадках порушень роботи технічного обладнання або програмного забезпечення негайно викликати представника технічної служби з питань експлуатації обчислювальної техніки;

у випадку виникнення різі в очах, різкого погіршення зору, виникнення головного болю, больових відчуттів у пальцях та кистях рук, посилення серцебиття – негайно припинити роботу з використанням ЕОМ, повідомити про те, що сталося, свого безпосереднього керівника й звернутися до медичної установи;

при загорянні обладнання негайно відключити його від електромережі;

про загорання повідомити свого безпосереднього керівника, оперативного чергового, пожежну службу; ужити заходів щодо ліквідації вогню за допомогою вуглекислотного або порошкового вогнегасника.

11. Засоби індивідуального захисту оператора від травмонебезпечних випромінювань оптичного діапазону, спінових, електромагнітних та інших полів ЕОМ з ВДТ і ПП

11.1. Згідно з ДСанПіН 3.3.2.007-98 для забезпечення захисту оператора та досягнення нормованих рівнів випромінювань ЕОМ з ВДТ і ПП рекомендовано застосування екранних фільтрів, локальних світлофільтрів (засоби індивідуального захисту очей) та інших засобів захисту, які пройшли випробування в акредитованих лабораторіях та отримали позитивний висновок державної санітарно-епідеміологічної експертизи.

11.2. Профілактичні заходи для зниження нервово-емоційного напруження зазначено в додатку 8 до ДСанПіН 3.3.2.007-98.

Проведення евакуації з приміщень і будівель:

1.1. Проведення організованої евакуації з виробничих та інших приміщень і будівель, запобігання проявам паніки і недопущення загибелі людей забезпечується шляхом:

– планування евакуації людей (складання плану евакуації з приміщення з розробленням схеми евакуаційних шляхів та виходів);

– визначення зон, придатних для розміщення евакуйованих з потенційно небезпечних зон;

– організації управління евакуацією;

– навчання населення діям під час проведення евакуації.

1.2. Працівники охорони в разі виявлення пожежі, спрацювання засобів пожежної сигналізації та автоматичного пожежогасіння повинні діяти за заздалегідь розробленою інструкцією, в якій визначаються їхні обов’язки з контролю за додержанням протипожежного режиму. Заступаючи на чергування, вони зобов’язані пересвідчитися в тому, що шляхи евакуації не захаращено, а двері евакуаційних виходів у разі потреби без перешкод відчиняються.

1.3. На магазині має бути встановлено порядок оповіщення людей про пожежу, з яким необхідно ознайомити всіх працівників.

1.4. Після оповіщення про пожежу до початку евакуації проходить певна затримка залежно від того, яку із систем оповіщення було використано для повідомлення про надзвичайну ситуацію (див. таблиця 4.1).

Таблиця 4.1 - Час затримки початку евакуації

Тип і характеристика будівлі	Час затримки початку евакуації, хв., при типах систем оповіщення		
	W1	W2	W3
Адміністративні, торговельні та виробничі будівлі (відвідувачі не сплять, знайомі з плануванням будівлі й процедурою евакуації)	<1	3	>4
Магазини, виставки, музеї, центри дозвілля та інші будівлі масового призначення (відвідувачі не сплять, але можуть бути не знайомі з плануванням будівлі й процедурою евакуації)	<2	3	>6
Гуртожитки, інтернати (відвідувачі можуть спати, але знайомі з плануванням будівлі й процедурою евакуації)	<2	4	>5
Готелі і пансіонати (відвідувачі можуть спати і бути не знайомі з плануванням будівлі й процедурою евакуації)	<2	4	>6
Госпіталі, будинки престарілих та інші подібні заклади (значне число відвідувачів може потребувати допомоги)	<3	5	>8

Примітка:

W1 — оповіщення та управління евакуацією оператором;

W2 — використання записаних заздалегідь типових фраз і інформаційних табло;

W3 — сирена пожежної сигналізації;

W4 — без оповіщення.

Узагальнюючи все наведене вище, можна зробити наступний висновок. Надзвичайно актуальним є створення й впровадження в практику розробки програмного забезпечення особистої зацікавленості як працівників так і роботодавців, у створення здорових і безпечних умов праці, що, в свою чергу, дозволить переорієнтувати працівників на безпечне виконання робіт.

Згідно діючому законодавству, роботодавець повинен створити для працівників безпечні, сприятливі умови праці. Однак, методи та способи організації роботи сучасного роботодавця, особливо на підприємствах недержавної форми власності, не дозволяють розглядати питання охорони праці як першочергові. Це питання залишається на другому місці, тому що, на перший погляд, не сприяють прибутковості організації. При цьому слід відзначити, що міжнародний досвід свідчить – організація праці, при якої інформуються вимоги безпеки і гігієни праці, підриває економічну ефективність установи і не може бути основою для стійкої стратегії розвитку.

Одним із шляхів розв’язання цієї проблеми є використання механізму колдоговірного регулювання питань охорони праці та захисту прав і інтересів працівників.

З метою поліпшення охорони праці в ІТ-компаніях до працівників можуть застосовуватися будь-які заохочення за активну участь та ініціативу у здійснені заходів щодо підвищення рівня безпеки та поліпшення умов праці. Види заохочень визначаються колективним договором, угодою.

ВИСНОВКИ

В результаті виконання магістерської роботи отримано наступні теоретичні та практичні результати:

1. Здійснено порівняльний аналіз відомих систем управління HR-процесами, виділено основні переваги та недоліки.

2. Обгрунтовано доцільність використання NoSQL систем в якості серверної частини для систем автоматизації HR-процесів, а також розкрито важливість процесів управління реплікаціями при оновленні записів.

3. Розроблено модель процесів узгодження реплік, що дозволяють розрахувати ймовірність читання застарілого запису з бази даних NoSQL для режимів синхронного і асинхронного поновлення запису.

4. Запропоновано модель процесу суворого узгодження реплік, яка дозволяє оцінити характеристики випадкового часу очікування початку читання записи з оновленої групи серверів.

5. Здійснено проектування інформаційної системи автоматизації HR-процесів із врахуванням сучасних тенденцій в галузі використання інформаційних технологій.

6. Здійснено програмну реалізацію інформаційної системи автоматизації HR-процесів з використанням Node.js та MongoDB, Node Express, який є веб-додатком для Node.js, а інтерфейс створений за допомогою HTML, Bootstrap, CSS та JS.

7. Проведено основні етапи тестування системи та її практичну апробацію, включаючи атапи встановлення, розгортання, налаштування, опис інструкції користувача.

8. Здійснено експериментальні дослідження моделей управління реплікаціями на етапі проектування системи автоматизації HR-процесів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Маклаков С.В. BPwin и ERwin: CASE-средства для разработки информационных систем. – М.: Диалог-Мифи, 1999. - 295 с.

2. Федорова Д.Э., Семенов Ю.Д., Чижик К.Н. CASE-технологии. - М.: Горячая линия Телеком, Радио и связь, 2005. – 160 с.

3. Самойлов, В. Д. Модельное конструирование компьютерных приложений / В. Д. Самойлов. - К.: Наукова думка. - 2007. - 198 с.

4. Лаврищева, Е. М. Методы программирования: теория, инженерия, практика / Е. М. Лаврищева. - К.: Наукова думка. - 2006. - 451 с.

5. Чмир, І.О. Моделювання систем у середовищі UML (Unified Modeling Language) : навч. посібник / І. О. Чмир, М. Ф. Ус ; Черкаськ. акад. менеджменту. - Черкаси : ЧАМ, 2004. - 100 с.

6. Андон, Ф. И. Логические модели интеллектуальных информационных систем / Ф. И. Андон, А. Е. Яшунин, В. А. Резниченко. - К: Наукова думка.-1999.- 397 с.

7. Ситник В.Ф. Системи підтримки прийняття рішень. – К.: Техніка, 2005. –164с.

8. Митчелл М., Оулдем Д., Самьюэл А. Программирование для Linux. Профессиональный поход. – М.: Вильямс, 2002. – 288 с.

9. Лапінський В. В., Габрусєв В. Ю. Основи операційних систем: Посібник для студентів. – К.: Вища школа, 2007. – 96 с.

10. Таненбаум Э., Вудхалл А. Операционные системы: разработка и реализация. Классика CS. – СПб.: Питер, 2006. – 576 с.

11. Codd, Edgar F.: A Relational Model of Data for Large Shared Data Banks. In: Communications of the ACM 13 (1970), June, No. 6, p. 377–387.

12. ACID. [Электронный ресурс] [http://ru.wikipedia.org/wiki/ACID]

13. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, Bigtable: a distributed storage system for structured data, Proceedings of the 7th Conference on USENIX Symposium on Operating

Systems Design and Implementation - Volume 7 (Seattle, WA, November 06 - 08, 2006), USENIX Association, Berkeley, CA, 15-15, 2006.

14. A.V. Burdakov, U.A. Grigorev, A.D. Ploutenko. Comparison of table join execution time for parallel DBMS and MapReduce, Software Engineering / 811: Parallel and Distributed Computing and Networks / 816: Artificial Intelligence and Applications Proceedings (March 18 – 18, 2014, Innsbruck, Austria), ACTA Press, 2014.

15. Bettina Kemme. Gustavo Alonso. Database Replication: a Tale of Research across Communities. Proceedings of the VLDB Endowment, Vol. 3, No. 1. P. 5-12.

16. Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Silberschatz, A., and Rasin, A. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. In Proceedings of the Conference on Very Large Databases, August 24-28, 2009, Lyon, France.

17. G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall and W. Vogels, Dynamo: Amazon's Highly Available Key-value Store, SOSP'07, October 14–17, 2007, Stevenson, Washington, USA, pp. 205-220, 2007.

18. Strauch, Ch. (2011), "NoSQL databases", Lecture Selected Topics on SoftwareTechnology Ultra-Large Scale Sites, Stuttgart Media University, p. 149, manuscript, available at: www. christof-strauch.de/nosqldbs.pdf (accessed 30 July 2012).

19. Gajendran, S.K. A survey on nosql databases // technical report, 2013, http://www.masters.dgtu.donetsk.ua/2013/fknt/babich/library/article10.pdf.

20. G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall и W. Vogels, «Dynamo: Amazon's Highly Available Key-value Store,» в 21st ACM Symposium on Operating Systems Principles, Stevenson, WA, 2007.

21. Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, David G. Andersen. Don't Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS // ACM SOSP'11, October 23-26, 2011, Cascais, Portugal. – P. 401-416.

22. M. Merideth and M. Reiter. Selected results from the latest decade of quorum systems research. In Replication, volume 5959 of LNCS, pages 185–206. Springer, 2010.

23. Цвященко Є.В. Аналіз адекватності моделі узгодження реплік в кінцевому рахунку в базах даних NosQL // Інформаційні технології. - 2015. - Т.21. № 11 - С. 840-848.

24. Aleksey Burdakov, Uriy Grigorev, Andrey Ploutenko, Eugene Ttsviashchenko "Estimation Models for NoSQL Database Consistency Characteristics", 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), 2016, pp. 35-42, doi: 10.1109/PDP.2016.23.

25. Крашівський А. І. Розробка веб-системи з використанням NODE.JS та MONGODB на прикладі системи автоматизації HR-процесів [Текст] / А. І. Крашівський // Комп’ютерні інформаційні технології : матеріали школи-семінару молодих вчених і студентів СІТ’2020 [м. Тернопіль, 30 листопада 2020 р.] / відп. за вип. М. П. Дивак. - Тернопіль : ЗУНУ, 2020. - С. 23-24.

ДОДАТКИ

Додаток А

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ
КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку проекту

«Веб клієнт електронної пошти, що буде підтримуватись мобільними пристроями для пошти tntu.edu.ua»

Розробники:
виконавець ст. гр. СПм-61
Крашівський А. І.

_____
(підпис)

керівник проекту
Михалик Дмитро Михайлович

_____
(підпис)

Тернопіль 2020

ЗМІСТ

1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану на 2020 рік, та згідно наказу на виконання дипломної роботи студента-магістра.

Тема проекту: «Розробка інформаційної системи автоматизації HR-процесів на базі Node.js».

2 ПРИЗНАЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Дипломна робота присвячена створенню системі автоматизації роботи з персоналом.

Предметом дослідження: є методи та програмні засоби реалізації інформаційної системи управління HR-процесами.

Мета роботи: розробка інформаційної системи автоматизації HR-процесів з використанням сучасних технології та нереляційних систем управління даними із врахуванням показників узгодження реплік в базах даних NoSQL на етапі проектування інформаційних систем.

3 ВИМОГИ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Функціональні вимоги

Система повинна передбачати чотири ролі:

- Адміністратор
- Бухгалтер
- Керівник проекту
- Працівник

Програмне забезпечення має виконувати наступні дії:

– управління привілеями користувачів;

– реєстрація працівників;

– перегляд профілю працівника;

– перегляд і зміна табелю відпрацьованого часу;

– видалення профілю користувача;
– затвердження заяви на відпустку;
– управління зарплатою працівників;
– формування заяви на відпустку;
– перегляд проектів;
– зарахування працівника на проект.

3.2 Технічні вимоги

Данна програмна система працює під будь-яким браузером так як це веб-сторінка яка запускається в браузері, вона має більшу привязку до браузера аніж до операційної системи.

3.3 Програмні вимоги

Програмний продукт повинен коректно функціонувати на мобільних пристроях та у браузерах. Розроблювана програмна система повинна бути пристосована для використання будь-яким користувачем. Розробку виконувати з використанням мови JavaScript та веб-сервером Node.js.

4. ЕТАПИ РОЗРОБКИ

В ходів реалізації роботи проект повинен пройти крізь наступні стадії розробки:

- аналіз предметної області;
- проектування архітектури;
- реалізація програмних модулів;
- тестування результатів розробки;
- оформлення супровідної документації;
- здача роботи.

5. ПРОГРАМНА ДОКУМЕНТАЦІЯ

Для програмного продукту повинні бути розроблені наступні документи:

- Пояснювальна записка;
- Технічне завдання;
- Презентаційний матеріал;
- Інструкція користувача;
- Додатки.

Пояснювальна записка до проекту оформляється згідно діючих вимог до нормоконтролю проектів.

6. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Розроблений програмний продукт має виконувати всі вимоги, що складаються з перерахованих у п. 3.1 характеристик.

Для здачі проекту необхідно підготувати весь перелік документів зазначений у п.5 цього документу.

Приймання проекту проводиться спеціально створеною комісією в термін зазначені в п.1 цього документу.

7. ВІДМІТКИ ПРО ВИКОНАННЯ ЕТАПІВ ТА ЗМІНИ В ПРОЕКТІ

Назва етапу	Відмітка*
Аналіз предметної області	
Архітектура системи	
Проектування веб-компонентів	
Використання системи	
Супровідна документація	

* відмітки про виконання етапу ставляться керівником проекту

ДОДАТОК Б

Програмний код

```
var express = require('express');
var router = express.Router();
var passport = require('passport');
var User = require('../models/user');
var Project = require('../models/project');
var csrf = require('csurf');
var csrfProtection = csrf();
var config_passport = require('../config/passport.js');
var moment = require('moment');
var Leave = require('../models/leave');
var Attendance = require('../models/attendance');


router.use('/', isLoggedIn, function isAuthenticated(req, res, next)
{
    next();
});

/**
 * Description:
 * Displays home page to the admin
 *
 *
 * Known Bugs: None
 */
router.get('/', function viewHome(req, res, next) {
    res.render('Admin/adminHome', {
        title: 'Admin Home',
        csrfToken: req.csrfToken(),
        userName: req.session.user.name
    });
});

/**
 * Description:
 * First it gets attributes of the logged in admin from the User
Schema.
 * Attributes are get with the help of id of logged in admin stored
in session.
 */

router.get('/view-profile', function viewProfile(req, res, next) {

    User.findById(req.session.user._id, function getUser(err, user)
{
        if (err) {
            console.log(err);
```

```
        }
        res.render('Admin/viewProfile', {
            title: 'Profile',
            csrfToken: req.csrfToken(),
            employee: user,
            moment: moment,
            userName: req.session.user.name
        });
    });

});

/**
 * Description:
 * Sorts the list of employees in User Schema.

 * Known Bugs: None
 */

router.get('/view-all-employees', function viewAllEmployees(req,
res, next) {

    var userChunks = [];
    var chunkSize = 3;
    //find is asynchronous function
    User.find({$or: [{type: 'employee'}, {type: 'project_manager'},
{type: 'accounts_manager'}]}).sort({_id: -1}).exec(function
getUsers(err, docs) {
        for (var i = 0; i < docs.length; i++) {
            userChunks.push(docs[i]);
        }
        res.render('Admin/viewAllEmployee', {
            title: 'All Employees',
            csrfToken: req.csrfToken(),
            users: userChunks,
            userName: req.session.user.name
        });
    });


});

/**
 * Description:
 * Displays add employee form to the admin.

 */

router.get('/add-employee', function addEmployee(req, res, next) {
    var messages = req.flash('error');
    var newUser = new User();

    res.render('Admin/addEmployee', {
```

```
        title: 'Add Employee',
        csrfToken: req.csrfToken(),
        user: config_passport.User,
        messages: messages,
        hasErrors: messages.length > 0,
        userName: req.session.user.name
    });

});

/**
 * Description:
 * First it gets the id of the given employee from the parameters.
 */
router.get('/all-employee-projects/:id', function
getAllEmployeePojects(req, res, next) {
    var employeeId = req.params.id;
    var projectChunks = [];

    //find is asynchronous function
    Project.find({employeeID: employeeId}).sort({_id: -
1}).exec(function findProjectOfEmployee(err, docs) {
        var hasProject = 0;
        if (docs.length > 0) {
            hasProject = 1;
        }
        for (var i = 0; i < docs.length; i++) {
            projectChunks.push(docs[i]);
        }
        User.findById(employeeId, function getUser(err, user) {
            if (err) {
                console.log(err);
            }
            res.render('Admin/employeeAllProjects', {
                title: 'List Of Employee Projects',
                hasProject: hasProject,
                projects: projectChunks,
                csrfToken: req.csrfToken(),
                user: user,
                userName: req.session.user.name
            });
        });

    });
});

/**

 */
router.get('/leave-applications', function getLeaveApplications(req,
res, next) {

    var leaveChunks = [];
```

```
    var employeeChunks = [];
    var temp;
    //find is asynchronous function
    Leave.find({}).sort({_id: -1}).exec(function findAllLeaves(err,
docs) {
        var hasLeave = 0;
        if (docs.length > 0) {
            hasLeave = 1;
        }
        for (var i = 0; i < docs.length; i++) {
            leaveChunks.push(docs[i])
        }
        for (var i = 0; i < leaveChunks.length; i++) {

            User.findById(leaveChunks[i].applicantID, function
getUser(err, user) {
                if (err) {
                    console.log(err);
                }
                employeeChunks.push(user);

            })
        }

        // call the rest of the code and have it execute after 3
seconds
        setTimeout(render_view, 900);
        function render_view() {
            res.render('Admin/allApplications', {
                title: 'List Of Leave Applications',
                csrfToken: req.csrfToken(),
                hasLeave: hasLeave,
                leaves: leaveChunks,
                employees: employeeChunks, moment: moment, userName:
req.session.user.name
            });
        }
    });

});

/**
 * Description:
 */
router.get('/respond-application/:leave_id/:employee_id', function
respondApplication(req, res, next) {
    var leaveID = req.params.leave_id;
    var employeeID = req.params.employee_id;
    Leave.findById(leaveID, function getLeave(err, leave) {

        if (err) {
            console.log(err);
        }
```

```
        User.findById(employeeID, function getUser(err, user) {
            if (err) {
                console.log(err);
            }
            res.render('Admin/applicationResponse', {
                title: 'Respond Leave Application',
                csrfToken: req.csrfToken(),
                leave: leave,
                employee: user,
                moment: moment, userName: req.session.user.name
            });


        })


    });


});


/**
 * Description:

 */
router.get('/employee-profile/:id', function getEmployeeProfile(req,
res, next) {
    var employeeId = req.params.id;
    User.findById(employeeId, function getUser(err, user) {
        if (err) {
            console.log(err);
        }
        res.render('Admin/employeeProfile', {
            title: 'Employee Profile',
            employee: user,
            csrfToken: req.csrfToken(),
            moment: moment,
            userName: req.session.user.name
        });

    });
});

/**
 * Description:

 */
router.get('/edit-employee/:id', function editEmployee(req, res,
next) {
    var employeeId = req.params.id;
    User.findById(employeeId, function getUser(err, user) {
        if (err) {
```

```
            res.redirect('/admin/');
        }
        res.render('Admin/editEmployee', {
            title: 'Edit Employee',
            csrfToken: req.csrfToken(),
            employee: user,
            moment: moment,
            message: '',
            userName: req.session.user.name
        });


    });

});

/**
 * Description:

 *
 * Known Bugs: None
 */
router.get('/edit-employee-project/:id', function
editEmployeeProject(req, res, next) {
    var projectId = req.params.id;
    Project.findById(projectId, function getProject(err, project) {
        if (err) {
            console.log(err);
        }
        res.render('Admin/editProject', {
            title: 'Edit Employee',
            csrfToken: req.csrfToken(),
            project: project,
            moment: moment,
            message: '',
            userName: req.session.user.name
        });


    });

});

/**
 * Description:
 * Gets the id of the employee from parameters.
 * Displays the add employee project form to the admin.
 *
 * Known Bugs: None
 */
router.get('/add-employee-project/:id', function
addEmployeeProject(req, res, next) {
```

```
    var employeeId = req.params.id;
    User.findById(employeeId, function getUser(err, user) {
        if (err) {
            res.redirect('/admin/');
        }
        res.render('Admin/addProject', {
            title: 'Add Employee Project',
            csrfToken: req.csrfToken(),
            employee: user,
            moment: moment,
            message: '',
            userName: req.session.user.name
        });

    });

});


/**
 * Description:
 * First finds project in the Project Schema with the help of id
from the parameters.
 * Gets the Employee of the project.

 *
 * Known Bugs: None
 */
router.get('/employee-project-info/:id', function
viewEmployeeProjectInfo(req, res, next) {
    var projectId = req.params.id;
    Project.findById(projectId, function getProject(err, project) {
        if (err) {
            console.log(err);
        }
        User.findById(project.employeeID, function getUser(err,
user) {
            if (err) {
                console.log(err);
            }
            res.render('Admin/projectInfo', {
                title: 'Employee Project Information',
                project: project,
                employee: user,
                moment: moment,
                message: '',
                userName: req.session.user.name,
                csrfToken: req.csrfToken()
            });
        })

    });
```

```
});

/**
 * Description:
 * Redirects admin to the employee profile page.

 * Known Bugs: None
 */
router.get('/redirect-employee-profile', function
viewEmployeeProfile(req, res, next) {
    var employeeId = req.user.id;
    User.findById(employeeId, function getUser(err, user) {
        if (err) {
            console.log(err);
        }
        res.redirect('/admin/employee-profile/' + employeeId);

    });

});

/**
 * Description:
 * Displays the admin its own attendance sheet

 * Known Bugs: None
 */
router.post('/view-attendance', function viewAttendance(req, res,
next) {
    var attendanceChunks = [];
    Attendance.find({
        employeeID: req.session.user._id,
        month: req.body.month,
        year: req.body.year
    }).sort({_id: -1}).exec(function viewAttendanceSheet(err, docs)
{
        var found = 0;
        if (docs.length > 0) {
            found = 1;
        }
        for (var i = 0; i < docs.length; i++) {
            attendanceChunks.push(docs[i]);
        }
        res.render('Admin/viewAttendanceSheet', {
            title: 'Attendance Sheet',
            month: req.body.month,
            csrfToken: req.csrfToken(),
            found: found,
            attendance: attendanceChunks,
            userName: req.session.user.name,
            moment: moment
        });
    });
```

```
});

/**
 * Description:
 * Known Bugs: None
 */
router.get('/view-attendance-current', function
viewCurrentlyMarkedAttendance(req, res, next) {
    var attendanceChunks = [];

    Attendance.find({
        employeeID: req.session.user._id,
        month: new Date().getMonth() + 1,
        year: new Date().getFullYear()
    }).sort({_id: -1}).exec(function getAttendanceSheet(err, docs) {
        var found = 0;
        if (docs.length > 0) {
            found = 1;
        }
        for (var i = 0; i < docs.length; i++) {
            attendanceChunks.push(docs[i]);
        }
        res.render('Admin/viewAttendanceSheet', {
            title: 'Attendance Sheet',
            month: new Date().getMonth() + 1,
            csrfToken: req.csrfToken(),
            found: found,
            attendance: attendanceChunks,
            moment: moment,
            userName: req.session.user.name
        });
    });

});

/**
 * Description:
 * Displays the attendance sheet of the given employee to the admin.
 */
router.get('/view-employee-attendance/:id', function
viewEmployeeAttendance(req, res, next) {
    var attendanceChunks = [];
    Attendance.find({employeeID: req.params.id}).sort({_id: -
1}).exec(function getAttendanceSheet(err, docs) {
        var found = 0;
        if (docs.length > 0) {
            found = 1;
        }
        for (var i = 0; i < docs.length; i++) {
            attendanceChunks.push(docs[i]);
        }
```

```
        User.findById(req.params.id, function getUser(err, user) {

            res.render('Admin/employeeAttendanceSheet', {
                title: 'Employee Attendance Sheet',
                month: req.body.month,
                csrfToken: req.csrfToken(),
                found: found,
                attendance: attendanceChunks,
                moment: moment,
                userName: req.session.user.name
                ,
                'employee_name': user.name

            })
        });
    });


});

/**
 * Description:

 */
router.post('/add-employee', passport.authenticate('local.add-
employee', {
    successRedirect: '/admin/redirect-employee-profile',
    failureRedirect: '/admin/add-employee',
    failureFlash: true,
}));

/**
 * Description:

 */
router.post('/respond-application', function respondApplication(req,
res) {
    Leave.findById(req.body.leave_id, function getLeave(err, leave)
{
        leave.adminResponse = req.body.status;
        leave.save(function saveLeave(err) {
            if (err) {
                console.log(err);
            }
            res.redirect('/admin/leave-applications');
        })
    })

});

router.post('/edit-employee/:id', function editEmployee(req, res) {
    var employeeId = req.params.id;
```

```
    var newUser = new User();
    newUser.email = req.body.email;
    if (req.body.designation == "Accounts Manager") {
        newUser.type = "accounts_manager";
    }
    else if (req.body.designation == "Project Manager") {
        newUser.type = "project_manager";
    }
    else {
        newUser.type = "employee";
    }
    newUser.name = req.body.name,
        newUser.dateOfBirth = new Date(req.body.DOB),
        newUser.contactNumber = req.body.number,
        newUser.department = req.body.department;
    newUser.Skills = req.body['skills[]'];
    newUser.designation = req.body.designation;

    User.findById(employeeId, function getUser(err, user) {
        if (err) {
            res.redirect('/admin/');
        }
        if (user.email != req.body.email) {
            User.findOne({'email': req.body.email}, function
getUser(err, user) {
                if (err) {
                    res.redirect('/admin/');
                }
                if (user) {
                    res.render('Admin/editEmployee', {
                        title: 'Edit Employee',
                        csrfToken: req.csrfToken(),
                        employee: newUser,
                        moment: moment,
                        message: 'Email is already in use',
userName: req.session.user.name
                    });
                }
            });
        }
        user.email = req.body.email;
        if (req.body.designation == "Accounts Manager") {
            user.type = "accounts_manager";
        }
        else if (req.body.designation == "Project Manager") {
            user.type = "project_manager";
        }
        else {
            user.type = "employee";
        }
        user.name = req.body.name,
            user.dateOfBirth = new Date(req.body.DOB),
            user.contactNumber = req.body.number,
```

```
            user.department = req.body.department;
        user.Skills = req.body['skills[]'];
        user.designation = req.body.designation;

        user.save(function saveUser(err) {
            if (err) {
                console.log(error);
            }
            res.redirect('/admin/employee-profile/' + employeeId);

        });
    });

});
router.post('/add-employee-project/:id', function
addEmployeeProject(req, res) {
    var newProject = new Project();
    newProject.employeeID = req.params.id;
    newProject.title = req.body.title;
    newProject.type = req.body.type;
    newProject.startDate = new Date(req.body.start_date),
        newProject.endDate = new Date(req.body.end_date),
        newProject.description = req.body.description,
        newProject.status = req.body.status;

    newProject.save(function saveProject(err) {
        if (err) {
            console.log(err);
        }
        res.redirect('/admin/employee-project-info/' +
newProject._id);
    });
});

router.post('/edit-employee-project/:id', function
editEmployeeProject(req, res) {
    var projectId = req.params.id;
    var newProject = new Project();

    Project.findById(projectId, function (err, project) {
        if (err) {
            console.log(err);
        }
        project.title = req.body.title;
        project.type = req.body.type;
        project.startDate = new Date(req.body.start_date),
            project.endDate = new Date(req.body.end_date),
            project.description = req.body.description,
            project.status = req.body.status;

        project.save(function saveProject(err) {
            if (err) {
                console.log(err);
```

```
            }
            res.redirect('/admin/employee-project-info/' +
projectId);

        });
    });

});


router.post('/delete-employee/:id', function deleteEmployee(req,
res) {
    var id = req.params.id;
    User.findByIdAndRemove({_id: id}, function deleteUser(err) {
        if (err) {
            console.log('unable to delete employee');
        }
        else {
            res.redirect('/admin/view-all-employees');
        }
    });
});

router.post('/mark-attendance', function markAttendance(req, res,
next) {

    Attendance.find({
        employeeID: req.session.user._id,
        date: new Date().getDate(),
        month: new Date().getMonth() + 1,
        year: new Date().getFullYear()
    }, function getAttendance(err, docs) {
        var found = 0;
        if (docs.length > 0) {
            found = 1;
        }
        else {
            var newAttendance = new Attendance();
            newAttendance.employeeID = req.session.user._id;
            newAttendance.year = new Date().getFullYear();
            newAttendance.month = new Date().getMonth() + 1;
            newAttendance.date = new Date().getDate();
            newAttendance.present = 1;
            newAttendance.save(function saveAttendance(err) {
                if (err) {
                    console.log(err);
                }

            });
        }
        res.redirect('/admin/view-attendance-current');
    });
```

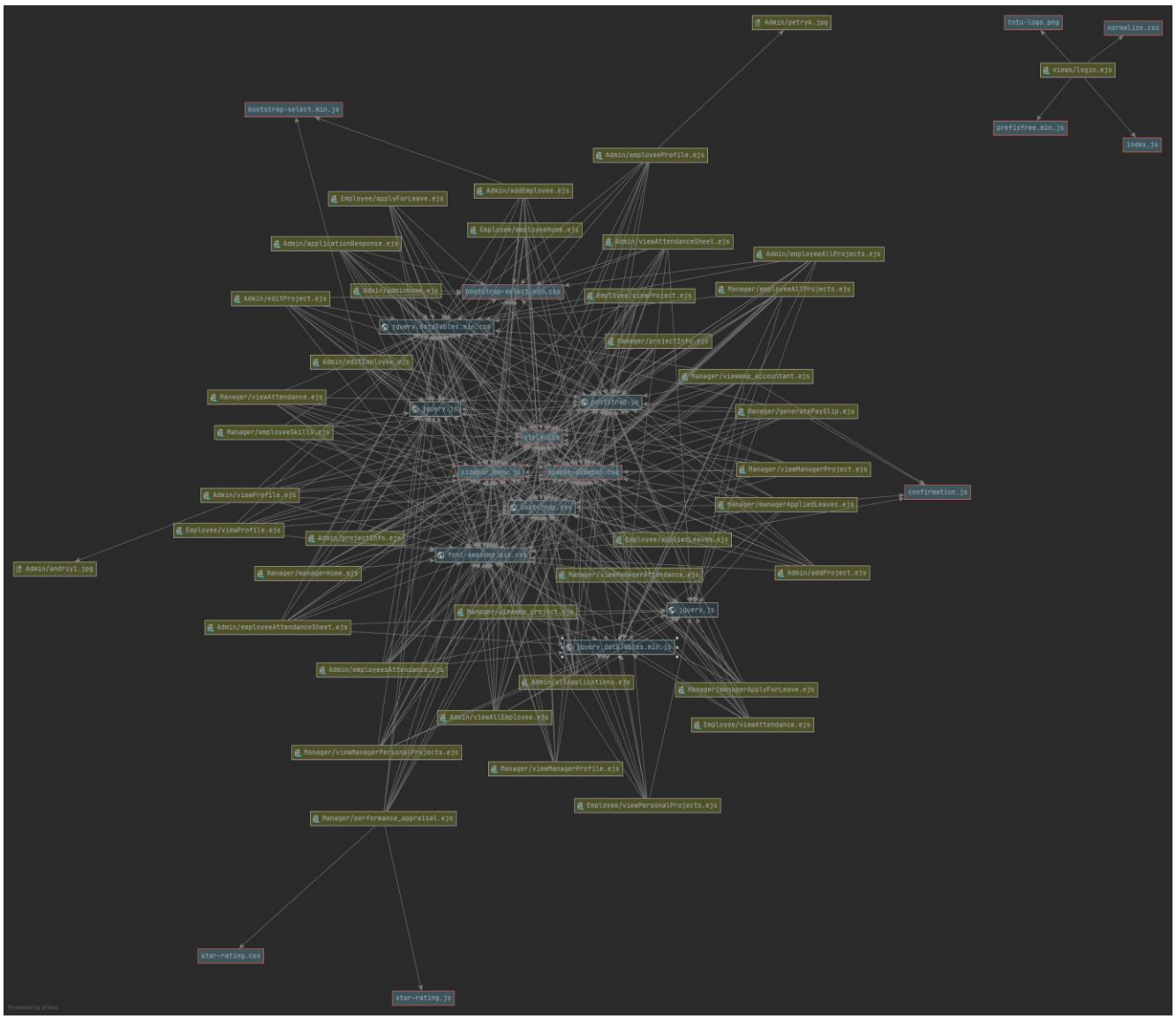
```
});
module.exports = router;

function isLoggedIn(req, res, next) {
    if (req.isAuthenticated()) {
        return next();
    }
    res.redirect('/');
}


function notLoggedIn(req, res, next) {
    if (!req.isAuthenticated()) {
        return next();
    }
    res.redirect('/');
}
```

ДОДАТОК В

Структура програмних модулів системи



ДОДАТОК Г

Слайди презентації

Розробка інформаційної системи автоматизації HR-процесів на базі Node.js

Крашівський Андрій Іванович

2020

Мета і наукова новизна

Метою роботи є розробка інформаційної системи автоматизації HR-процесів з використанням сучасних технології та нереляційних систем управління даними із врахуванням показників узгодження реплік в базах даних NoSQL на етапі проектування інформаційних систем.

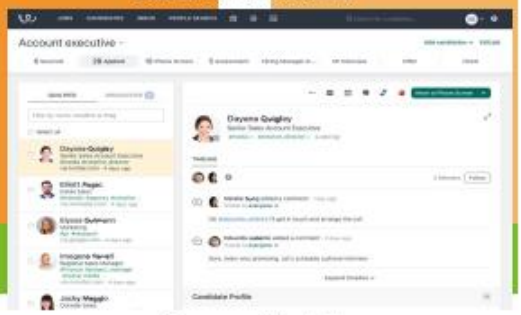
Об'єкт дослідження – процеси реалізації інформаційної системи управління HR-процесами.

Предмет дослідження методи та програмні засоби реалізації інформаційної системи управління HR-процесами.

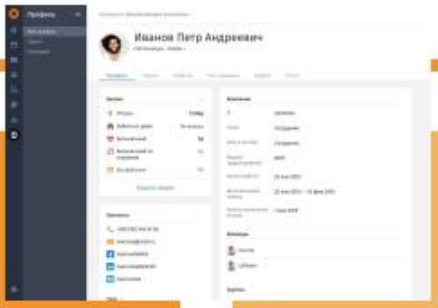
Наукова новизна одержаних результатів. Запропоновано модель процесів узгодження реплікацій, що дозволяють розрахувати ймовірність зчитування застарілого запису з бази даних NoSQL для режимів синхронного і асинхронного оновлення даних.

Аналіз відомих HR-систем

Система BambooHR



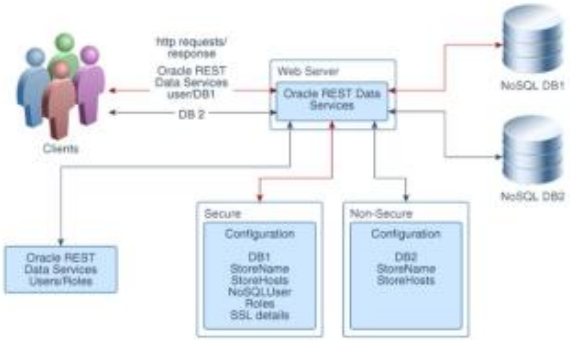
Система Workable



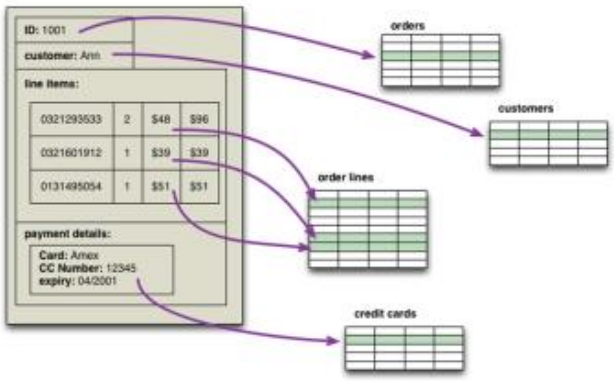
Система Hurma System

Система Zoho People

Особливості використання NoSQL систем

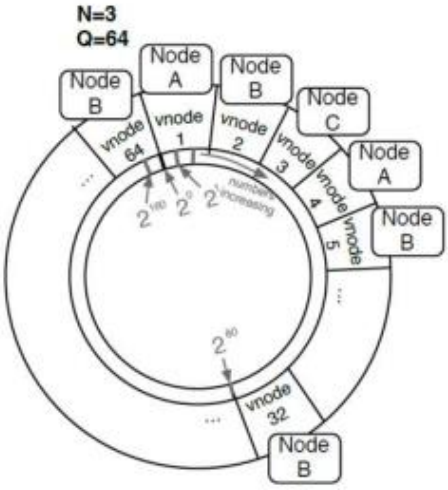


Доступ до бази даних NoSQL по протоколу REST

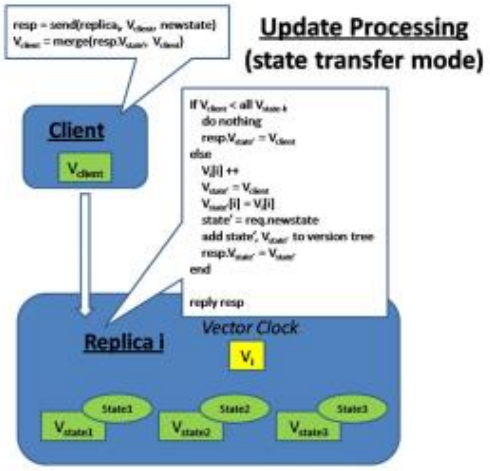


Зберігання запису в базі даних

Функції узгодження реплік в базах даних NoSQL

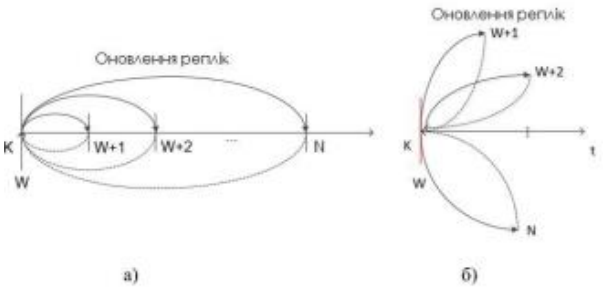


Кільце з 64 v-вузлами і трьома серверами

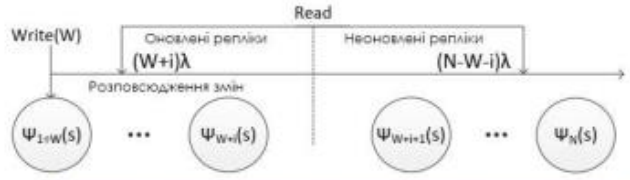


Приклад ведення вектора годин

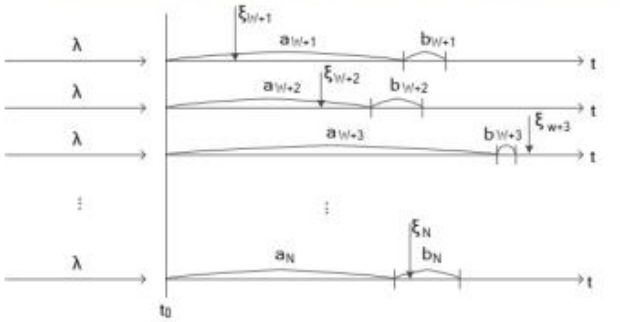
Модель процесів узгодження реплік при оновленні будь-якого запису бази даних



Варіанти поширення змін: а) синхронний режим; б) асинхронний режим.



Модель узгодження реплік в кінцевому рахунку ($W + R \leq N$, синхронний режим)



Модель узгодження реплік в кінцевому рахунку ($W + R \leq N$, асинхронний режим)

Алгоритми процесу управління записами

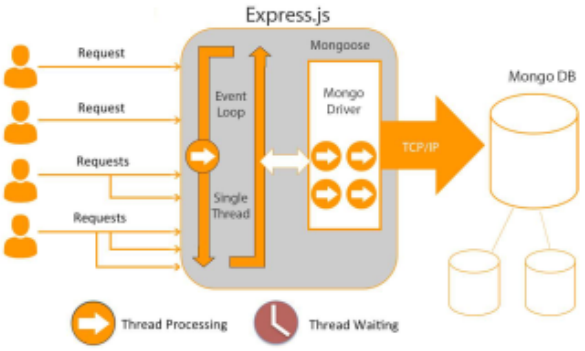
Алгоритм:
REC_VAL = 1
ЦИКЛ по N_ITER_W
  TIME_STAMP = CURR_TIME
  ЗАПИСАТИ в БД запис <KEY, REC_VAL>
  ЗАПИСАТИ в журнал <CURR_TIME, REC_VAL>
  REC_VAL += 1
  DELAY = EXPONENTIAL(1.25)
  DELAY -= (CURR_TIME - TIME_STAMP)
  ЗАТРИМАТИ виконання на час DELAY
КІНЕЦЬ ЦИКЛУ

Алгоритм процесу оновлення записів

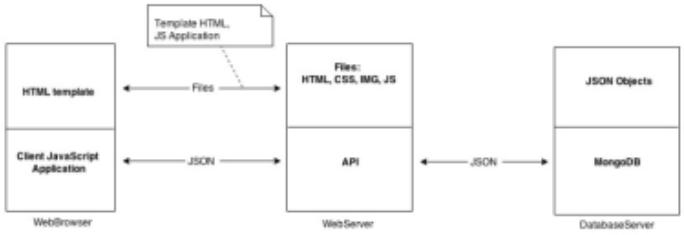
Алгоритм:
N_ITER_R = N_ITER_W / 1.25 · λ
ЦИКЛ по N_ITER_R
  TIME_STAMP = CURR_TIME
  ЗЧИТАТИ з БД запис <KEY, REC_VAL>
  ЗАПИСАТИ в журнал <CURR_TIME, REC_VAL>
  DELAY = EXPONENTIAL(λ)
  DELAY -= (CURR_TIME - TIME_STAMP)
  ЗАТРИМАТИ виконання на час DELAY
КІНЕЦЬ ЦИКЛУ

Алгоритм процесу зчитування записів

Архітектура системи

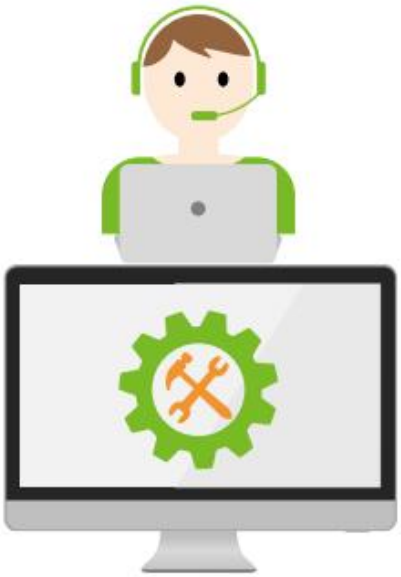


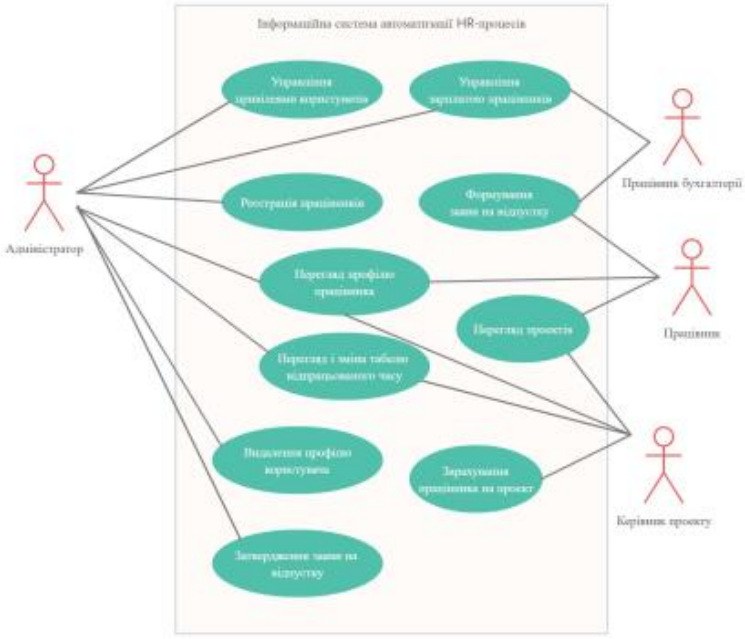
Діаграма розгортання системи



Обмін даними в форматі JSON в трирівневій архітектурі

Діаграма варіантів використання інформаційної системи автоматизації HR-процесів





Структурні компоненти системи автоматизації HR-процесів

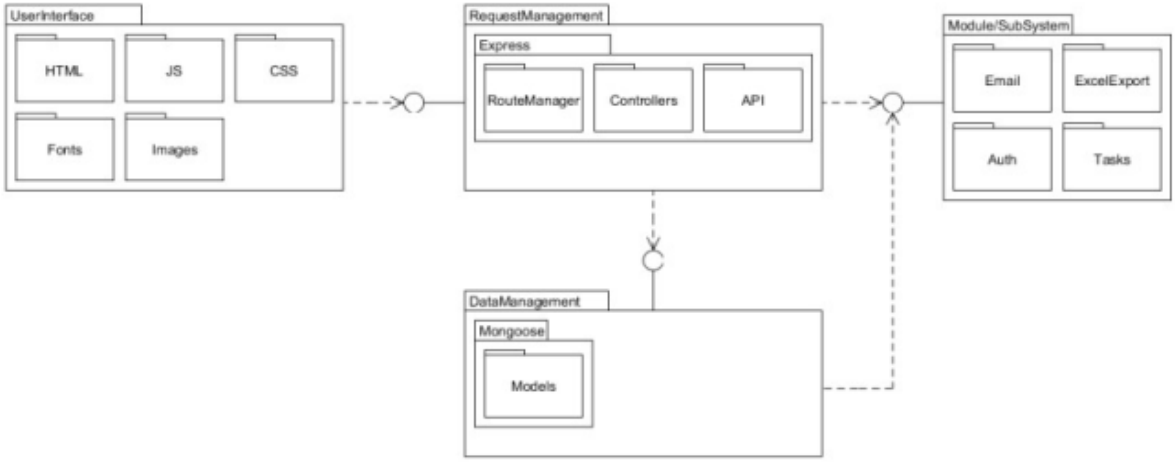
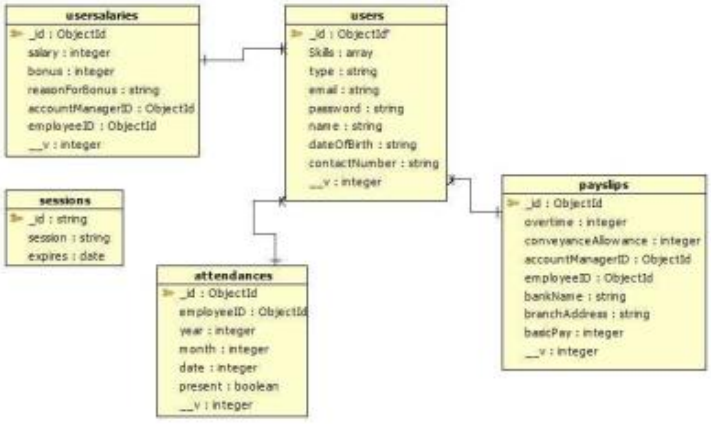


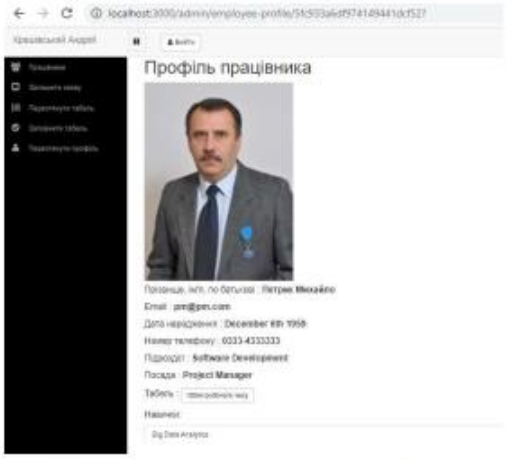
Схема бази даних з позначенням псевдо-зв'язків в MongoDB



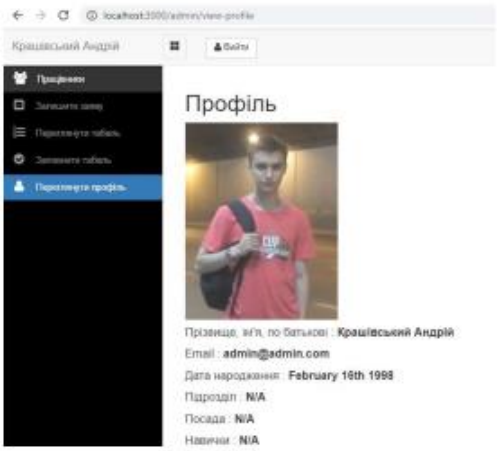
Адміністратор

Бухгалтер

Реалізація функціоналу з інформацією про працівника

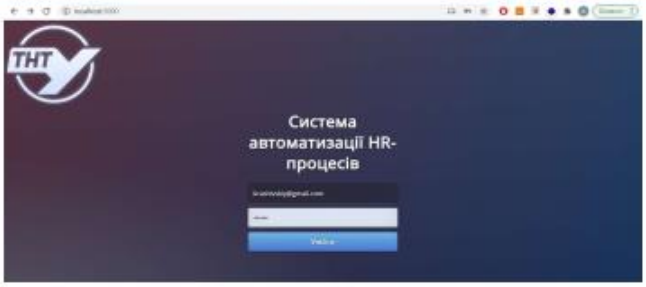


Сторінка відображення інформації про працівника



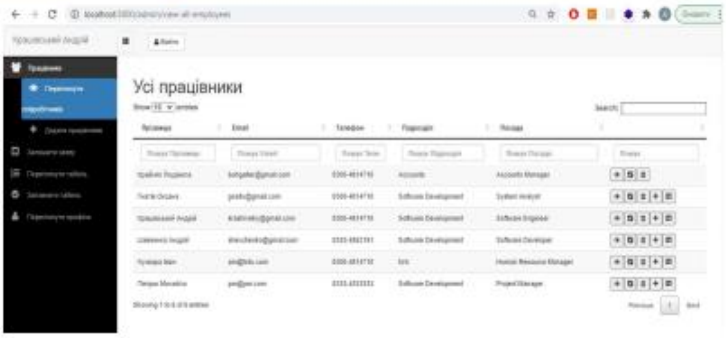
Сторінка профілю адміністратора

Реалізація функціоналу з інформацією про працівника

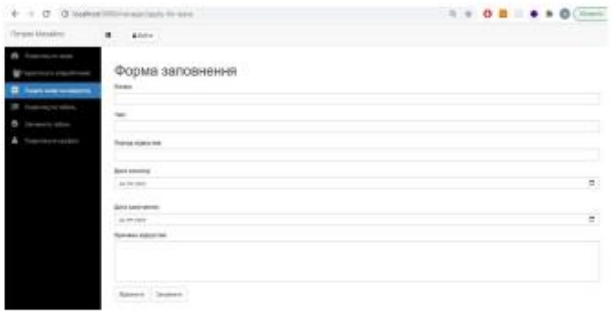


Головна сторінка для входу в систему

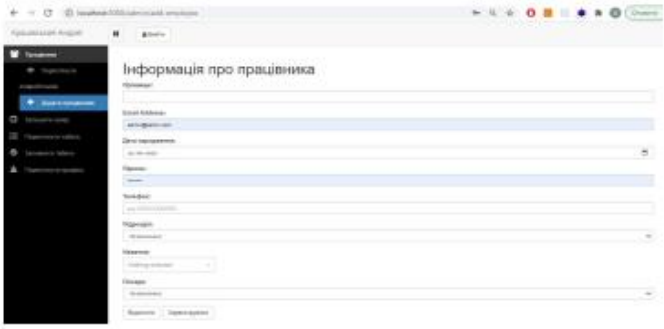
Сторінка відображення інформації про працівників



Реалізація функціоналу

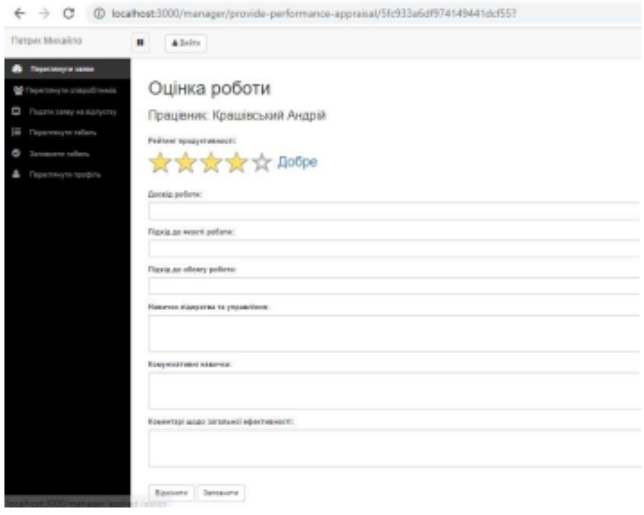


Форма подачі заяви на відпустку

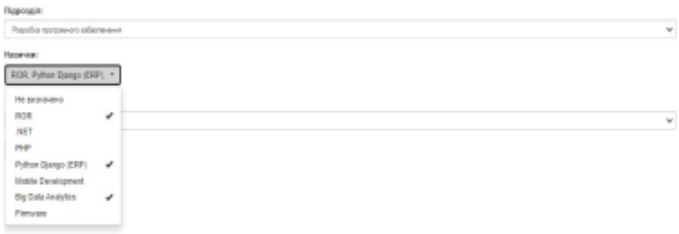


Форма реєстрації нового працівника

Реалізація функціоналу оцінки роботи

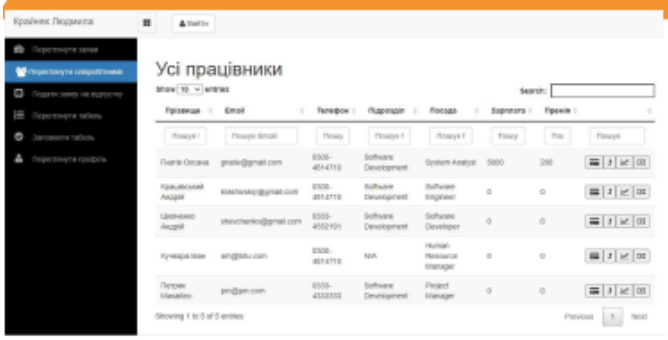


Форма оцінки результативності роботи працівників

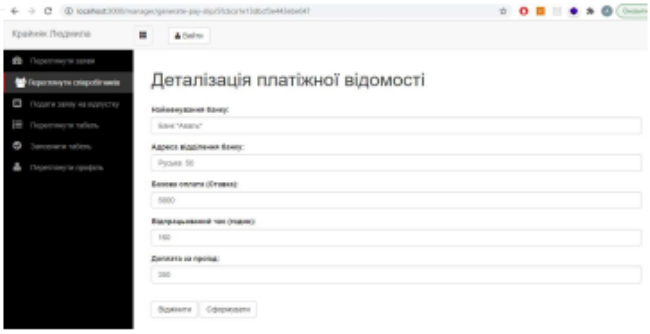


Форма відзначення практичних навичок

Реалізація функціоналу оплати праці

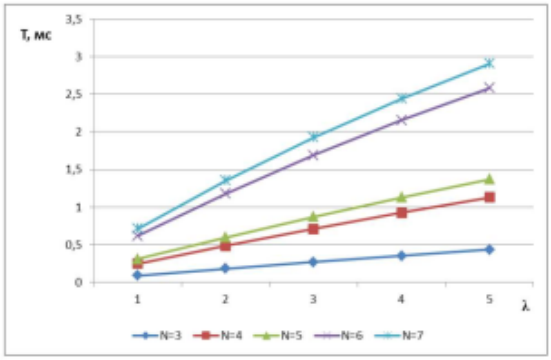


Сторінка обліку оплати праці співробітників

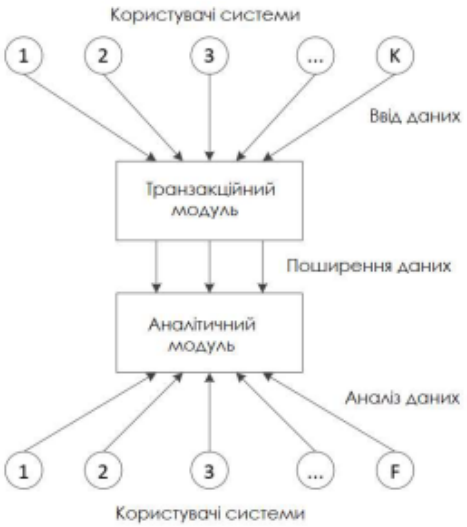


Сторінка формування платіжної відомості для окремого працівника

Експериментальні дослідження



Залежності середнього часу очікування T (мс) від інтенсивності надходження вимог на читання λ (1/с) при різних значеннях числа реплік запису N



Структура модулів системи управління HR-процесами на етапі проектування

Висновки

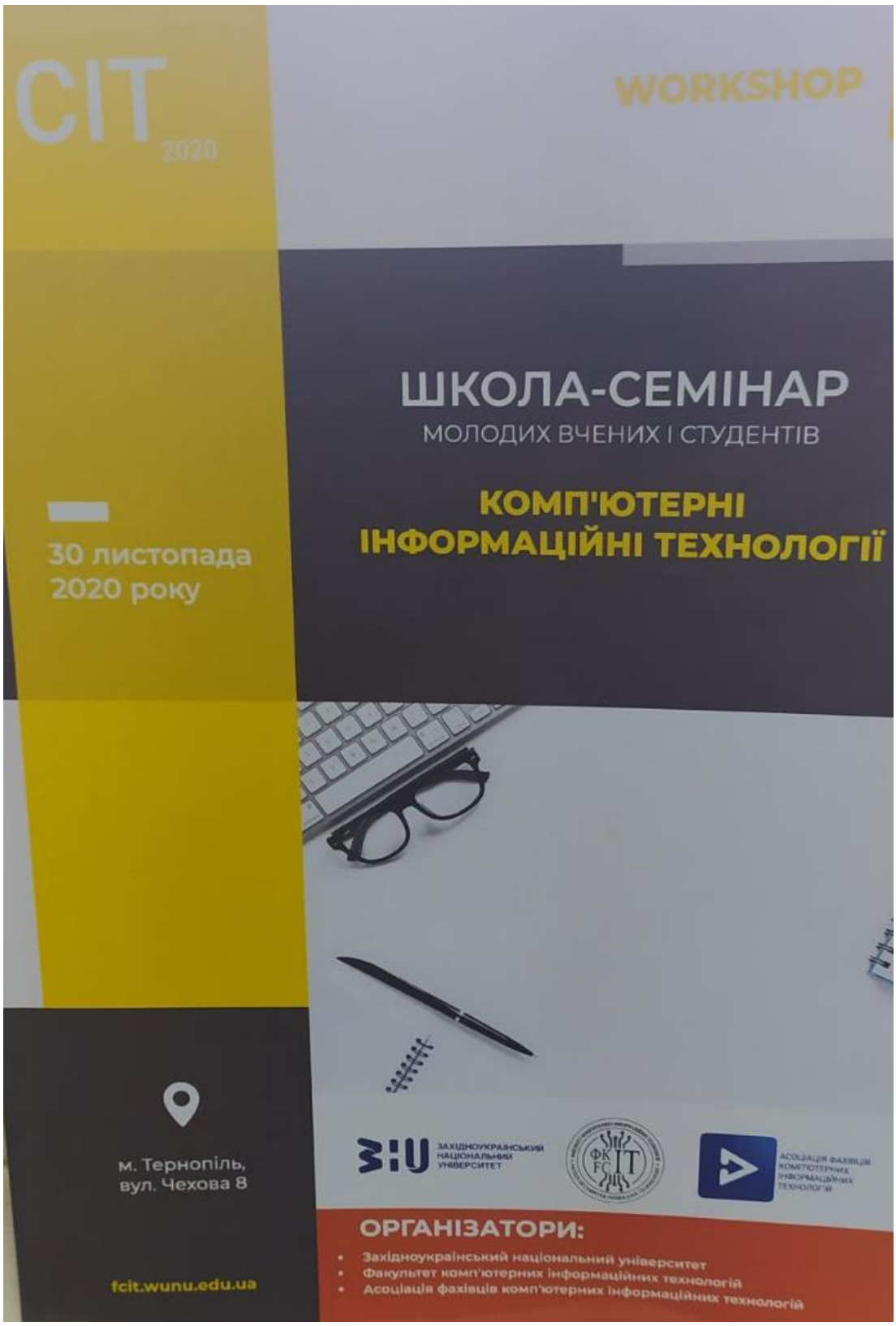
В результаті виконання магістерської роботи отримано наступні теоретичні та практичні результати:

1. Здійснено порівняльний аналіз відомих систем управління HR-процесами, виділено основні переваги та недоліки.
2. Обгрунтовано доцільність використання NoSQL систем в якості серверної частини для систем автоматизації HR-процесів, а також розкрито важливість процесів управління реплікаціями при оновленні записів.
3. Розроблено модель процесів узгодження реплік, що дозволяють розрахувати ймовірність читання застарілого запису з бази даних NoSQL для режимів синхронного і асинхронного поновлення запису.
4. Здійснено проектування інформаційної системи автоматизації HR-процесів із врахуванням сучасних тенденцій в галузі використання інформаційних технологій.
5. Здійснено програмну реалізацію інформаційної системи автоматизації HR-процесів з використанням Node.js та MongoDB, Node Express, який є веб-додатком для Node.js, а інтерфейс створений за допомогою HTML, Bootstrap, CSS та JS.
6. Проведено основні етапи тестування системи та її практичну апробацію, включаючи атапи встановлення, розгортання, налаштування, опис інструкції користувача.
7. Здійснено експериментальні дослідження моделей управління реплікаціями на етапі проектування системи автоматизації HR-процесів.

Дякую!

ДОДАТОК Д

Апробація результатів роботи



ЗМІСТ

УДК 004.9

РОЗРОБКА ВЕБ-СИСТЕМИ З ВИКОРИСТАННЯМ NODE.JS ТА MONGODB НА ПРИКЛАДІ СИСТЕМИ АВТОМАТИЗАЦІЇ HR-ПРОЦЕСІВ

Крашівський А.І.
Тернопільський національний технічний університет імені Івана Пулюя, магістрант

I. Постановка проблеми

Дана робота присвячена процесу проектування, розробки та розгортання на віддаленому сервері веб-додатку для автоматизації HR-процесів з використанням фреймворку Angular.js, платформи Node.js і системою управління базами даних MongoDB. Цей додаток повинен зберігати інформацію про людські ресурси, освіту всіх співробітників, їх останній досвід, відвідуваність, управління відпустками та поточне управління проектами. В рамках даної системи повинна бути реалізована адміністративна панель для управління всією інформацією про працівників. Система міститиме статистичні дані про ставку найманих працівників з різних установ та середню плинність працівників з різних установ. Особлива увага приділена огляду принципів роботи, основних можливостей і переваг використаних інструментів і технологій.

II. Мета роботи

Метою дослідження є розробка інформаційної системи автоматизації HR-процесів на базі Node.js та MongoDB.

III. Вибір програмних інструментів для реалізації системи

Для вирішення поставлених завдань було вирішено створити SPA (Single Page Application - веб-сайт або веб-сайт, що використовує єдиний HTML-документ як оболочку для всіх веб-сайтів та організує взаємодію з користувачем за допомогою динамічно завантаженого HTML,CSS, JavaScript без перезагрузки всієї сторінки) [1]. Для створення сучасних SPA на стороні клієнта існує багато фреймворків, одним з яких є Angular.js, ми його використовуємо для клієнтської частини нашого додатку, надає більші можливості для реалізації складних веб-додатків. Основними можливостями є зв'язування даних моделейта презентації, розбиття додатків на модулі, підтримка шаблонів, функції для зручної роботи з promise, http-запитами.

Клієнт додатку на етапі розробки представляє сукупність розділених CSS, HTML, JSфайлів, щоб мати можливість використовувати їх усі разом необхідна система збору фронтенда, ми будемо використовувати Webpack, за допомогою його ми зможемо зібрати всі js-файли в одному бандлі,використовуємо css препроцесори, такі як Sass, і так вже у нас з'являється можливість для модульного розбиття html і підключення його в js функцією require().

Для написання серверної логіки використовується Node.js - програмна платформу,заснована на движку V8 (транслятор JavaScript в машинний код). Node.js надає можливість JavaScript взаємодіяти з пристроями вводу-виводу через свій API (написаний на C++), підключити інші зовнішні бібліотеки, написані на різних мовах, забезпечують виклики до них із JavaScript-коду [2]. Також Node.js розміщується разом із пакетним менеджером NPM за допомогою якого ви можетешвидко отримати доступ до модулів під різні потреби. Використовуючи NPM можназручно керувати всіма зовнішніми залежностями додатків, надаючи можливість автоматичного завантаження їх у каталог node_modules у папку проекту за допомогою команд npm install.

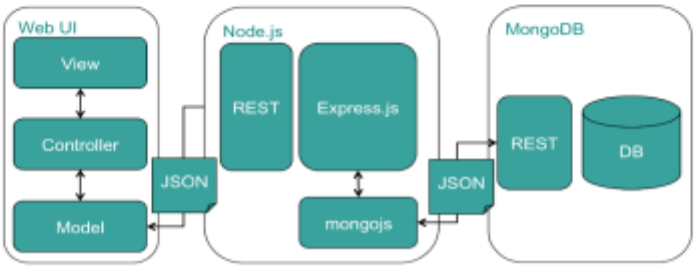


Рисунок 1–Архітектура системи

Серед переваг Node.js можна виділити асинхронність, неблокуючий ввід-вивід, можливість написання ізоморфних додатків, використовуючи один код на клієнтах і на серверах. Також нам потрібна база даних, вибір здійснено на користь MongoDB - документована база даних NoSQL даних, яка зберігає інформацію в документах JSON. У порівняні з реляційними базами дані документованих БД мають перевагу в швидкості роботи та менше займають пам'яті. Для аутентифікації користувачів буде використовуватися концепція JWT (JSON Web Token).Для спілкування між клієнтами та сервером ми будемо використовувати REST - це стиль архітектури програмного забезпечення, при якому сервер не зберігає стан клієнта, а також кожен запит від клієнта до сервера містить у собі вичерпну інформацію про бажаний доступ до сервера. Дії над даними завданнями виконуються за допомогою методів: GET (отримати), PUT (додати, замінити),POST (додати, змінити, видалити), DELETE (видалити). Таким чином, дії CRUD (Create-Read-Updtae-Delete) можуть виконуватися як з усіма 4-ма методами, так і лише за допомогою GET і POST. Використання методології REST забезпечує переваги за рахунок простоти, масштабності, надійності та продуктивності сервісів.

III. Реалізація системи

Для підтримки масштабованості, зручності внесення змін і збереження чіткої логічної структури всіх модулів клієнтської частини нашого додатку дуже важливо правильно вибрати метод розбиття файлів на директорії і піддиректорії (див. рис. 2).

а) б)

Рисунок 2–Структура каталогу файлів клієнтської частини (а) та серверної частини (б) системи

Для того щоб розгорнути нашу систему в інтернеті було вирішено скористатися безкоштовними функціями хмарної інтернет-платформи Heroku, яка крім Node.js підтримує також мови Java, Ruby, Scala, PHP та інші. Програми, що працюють на Heroku, використовують також DNS-сервер Heroku(зазвичай додатки мають доменне ім'я виду «ім'я_додатку.herokuapp.com»). Для кожного додатку виділяється кілька незалежних віртуальних процесів, які називаються «dynos».Вони розподілені по спеціальній віртуальній сітці («dynos grid»), яка складається з декількох серверів. Крім цього є зручна можливість налаштувати автоскладання додатку з git-репозиторію [2]. В результаті розроблена система була успішно розміщена і тепер доступна по домену. Система планується до практичного використання в одній з IT компаній міста Тернопіль для управління HR процесами.

Висновок

В рамках проведеного дослідження реалізована веб-система управління HR-процесами з використанням сучасних інформаційних технологій. Здійснено апробацію системи на прикладі автоматизації основних HR бізнес процесів водній з провідних компаній Тернополя.

Список використаних джерел

1. Міковскі Майкл, Пауелл Джош. Розробка односторінкових веб-додатків // ДМК Пресс,2014. 512 с.
2. Lambert M. Surhone, Mariam T. Tennoe, and Susan F. Henssonow. 2010. Node.js. Betascript Publishing, Beau Bassin, MUS.