МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет магістерської та аспірантської підготовки
Кафедра інформаційних технологій

КОМПЛЕКСНА МАГІСТЕРСЬКА РОБОТА

на тему: «Розробка системи виявлення плагіату в наукових та студентських роботах»

Склад:

Частина 1. «Серверна частина»

Виконавець: Федючек О. В.
(П.І.Б.)

Керівник: Терещенко Т.М.
(П.І.Б.)

Частина 2. «Клієнтська частина»

Виконавець: Щекотілін В.А
(П.І.Б.)

Керівник: Терещенко Т.М.
(П.І.Б.)

Староста роботи: Щекотілін В.А
(П.І.Б.)

Провідний керівник проекту: к.т.н., доц. Терещенко Т.М.
(П.І.Б.)

Рецензент: к.т.н., доц. Гнатовська Г.А.
(П.І.Б.)

ОДЕСА 2017

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет магістерської та
аспірантської підготовки
Кафедра інформаційних
технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Розробка клієнтської частини

Виконав студент 2 курсу групи МК-61
спеціальності 8.05010101 Інформаційні
управляючі системи та технології,
Щекотілін Вадим Андрійович

Керівник к.т.н., доцент
Терещенко Тетяна Михайлівна

Рецензент к.т.н., доцент
Гнатовська Ганна Арнольдівна

Одеса 2017

АНОТАЦІЯ

на магістерську роботу студента гр. МК-61 Щекотіліна В.А.

Тема магістерської роботи «Система виявлення плагіату в наукових та студентських роботах».

Актуальність даної магістерської роботи полягає в необхідності захисту інтелектуальної власності.

Об'єкт дослідження – Система виявлення плагіату в наукових та студентських роботах.

Мета роботи – створення односторінкового додатку, основною функцією якого є виявлення відсотку схожості між вмістом завантажуванного файла і файлами, які знаходяться у локальному сховищі.

Представлена магістерська робота містить пояснювальної записки – 63 сторінки, малюнків – 32, таблиць – 0, посилань – 15.

Ключові слова: SPA, плагіат, пошук плагіату

ANNOTATION

Master work "system detect plagiarism in academic and student works."

The relevance of this master's work is the need to protect intellectual property.

The object of study - the system detect plagiarism in academic and student work.

Purpose - to create single-application whose main function is to identify the percentage of similarity between the content downloading files and files that are in the local repository.

Presented master thesis contains explanatory notes - 63 pages, pictures – 32, tables – 0, references - 15.

Keywords: SPA, plagiarism, plagiarism search

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

Скорочення

CSS – Cascading Style Sheets – каскадні таблиці стилів

HTML – HyperText Markup Language – мова гіпертекстової розмітки

БЄМ – Методологія

Терміни

Аккаунт – обліковий запис, де зберігається персональна інформація користувача для входу на сайт

Хостинг – послуга з надання простору для розміщення сайтів в мережі Інтернет

NodeJS – вільний веб-сервер

MongoDB – система управління базами даних (СУБД).

ВСТУП

Актуальність систем виявлення плагіату дуже велика. На сьогоднішній день плагіат можливо побачити у майже всіх сферах людської діяльності, перш за все причиною такої ситуації стала неймовірна популярність глобальної мережі «Інтернет», яка дозволяє публікувати на сторінках абсолютно все, що бажає користувач, не перевіряючи це на авторські права. Захист права інтелектуальної власності в Україні останніми роками набуває усе більшої гостроти, оскільки масштаби порушення цих прав стрімко зростають. Варто хоча б згадати про проблему виробництва та реалізації неліцензійних компакт-дисків, відеокасет, комп'ютерних програм тощо, а також пов'язані з цим ускладнення міжнародних торговельних відносин у цій та суміжній сферах.

Основною метою магістерської роботи є розробка системи виявлення плагіату.

Під час виконання магістерскької роботи було виконано низку задач. Перш за все був проведений аналіз предметної області, де було проаналізовано анологічні системи та поставлені цілі щодо системи. Наступним кроком була вибрана архітектура системи, та допоміжні технічні засоби для написання программи. На третьму єтапі було спроектовано систему, за допомогою ряду технологій проектування. І останнім кроком було реалізовано систему.

Результати роботи були докладені мною, у минулому році, на конференції.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Дослідження предмета, цілей і особливостей систем виявлення плагіату

На сьогоднішній день значно підвищилися вимоги до оригінальності змісту не тільки курсових і дипломних робіт, дисертації і докторські праці викладачів в цьому плані також перевіряються досить строго. Саме тому так важливо використовувати програми плагіату тексту, які дозволяють перевірити унікальність і виявити збіги, а в подальшому і допомагають в редагуванні.

Утиліти, які перевіряють оригінальність, визначають збіги: якщо в вашому творі вставлені невиправлені і раніше використані фрагменти.

У плагіату існує безліч визначень, однакових з точністю до формулювань. У даній роботі під плагіатом, як і всюди, мається на увазі умисне привласнення авторства чужого роботи або її частини. Відповідно, плагіат знаходиться в сфері дії авторського права. Слід зазначити, що можливий і «ненавмисний плагіат», який найбільш відомий по музичним творам. Складність проблеми останнього полягає в можливу дію об'єктивних чинників – у властивостях і / або розладах людської пам'яті (наприклад, кріптомнезіі – «забування джерела» інформації), стилізації – підгоні твори під прийняті в даній галузі норми, банальних збіги, перетворює проблему визначення плагіату в окремих випадках в нетривіальну

Таким чином, плагіат в будь-якому разі розглядається як шахрайство, суть якого – у крадіжці чужої роботи або її частини і представленні її як власної. Загалом, плагіат можна поділити на три основні типи:

– Копіювання чужої роботи та оприлюднення її під своїм іменем.

– Представлення суміші власних та запозичених в інших аргументів без належного цитування джерел.

– Перефразування чужої роботи без належно оформленого посилання на оригінального автора або видавця.

1.2 Аналіз аналогічних систем

Для визначення вимог, функцій, бізнес-логіки розроблюваної спеціалізованої інформаційної системи виявлення плагіату проведений

необхідний аналіз існуючих і функціонуючих в мережі Інтернет аналогічних систем. У ході аналізу були визначені основні переваги та недоліки таких систем.

Відмінності між такими системами визначаються за наявністю різноманітних додаткових способів пошуку, якості пошуку, наявності генерації звітів, ціни.

Розглянемо деякі спеціалізовані інформаційні системи антиплагіату.

PlagScan – поширена система пошуку плагіату. Базується у США, та має велику кількість клієнтів від звичайних людей, до великих компаній та університетів. (рис. 1.1)[1]

Рисунок 1.1 – Головна сторінка системи «PlagScan»

Ця система має приємний дизайн сайту, великий вибір планів користування системою, різноманітні знижки. Можна сказати, що ця компанія має дуже великий досвід у використанні подібних систем. Також вони надають багато різноманітних пакетів послуг для юридичних та фізичних осіб.

Перевагою даної системи є:

– велика база даних текстів;
– доступ до тисяч організацій та їх сховищ даних;
– різноманіття планів та цін на послуги для різних людей.

Недоліками є:

– лише на англійській мові;
– приватна система (приватні алгоритми пошуку);
– висока вартість.

Наступна розглянута система – це Unplag.com[2]. Вона майже ідентична першій системі, але має українське коріння (рис. 1.2).

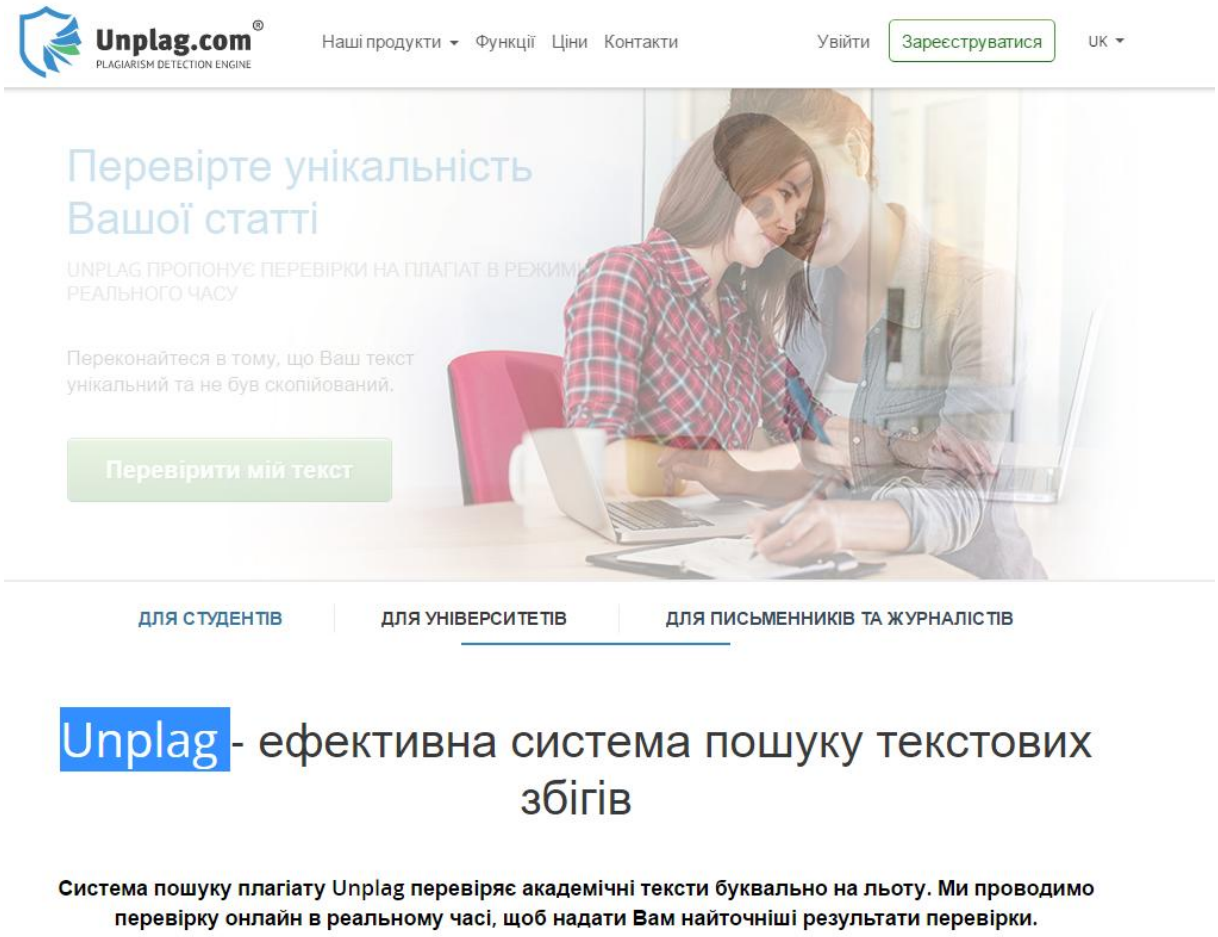

Рисунок 1.2 – Головна сторінка сайту системи «Unplag.com»

Вони позиціонують себе як онлайн-сервіс пошуку плагіату, який перевіряє текстові документи на наявність запозичених частин тексту з відкритих джерел в Інтернеті чи внутрішньої бази документів користувача

Перевагою даної системи є:

– зручність у користуванні;
– різноманітні плани користування;
– декілька способів пошуку плагіату;

– доступність десятка мов для текстів;
– створення звітності за результатами пошуку;
– розпізнавання підміни символів у тексті.

Недоліками є:
– приватність внутрішнього алгоритму пошуку;
– вартість.

Наступна розглянута система – це Антиплагиат[3] (рис. 1.3).

.

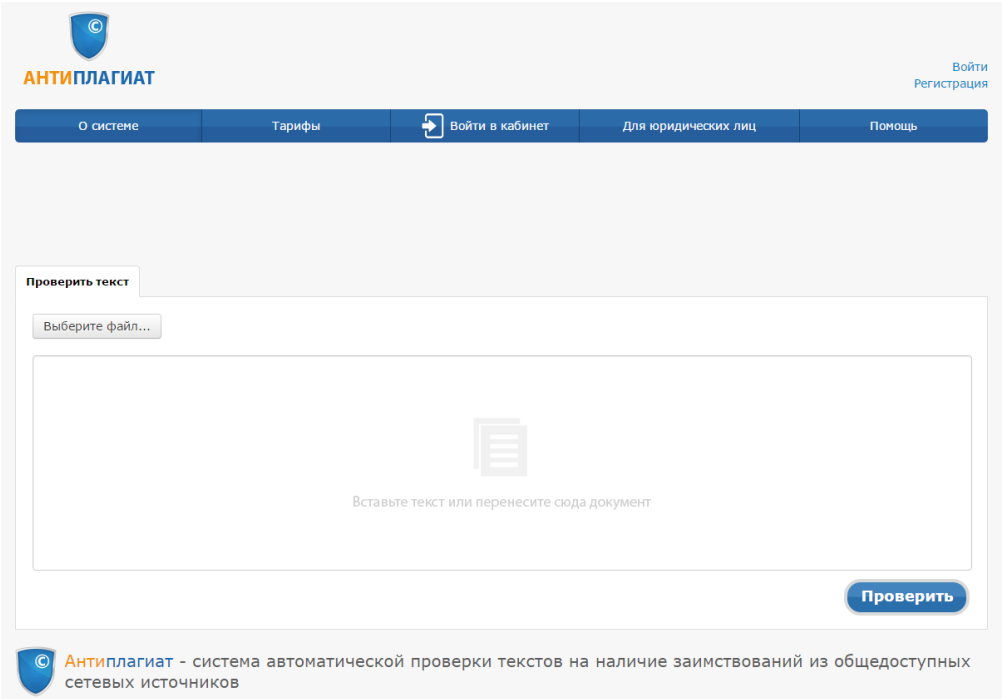

Рисунок 1.3 – Головна сторінка сайту системи «Антиплагиат»

Ця система має не дуже приємний дизайн сайту та не такий великий функціонал як інші системи.

Перевагою даної системи є:
– зручність у користуванні;
– простота системи;
– можливість роботи без реєстрації.

Недоліками є:
– один спосіб пошуку плагіату;
– відсутність планів для фізичних або юридичних осіб.

Наступна розглянута система – це Content-watch. За функціоналом подібна до Антиплагіату. (рис. 1.4)

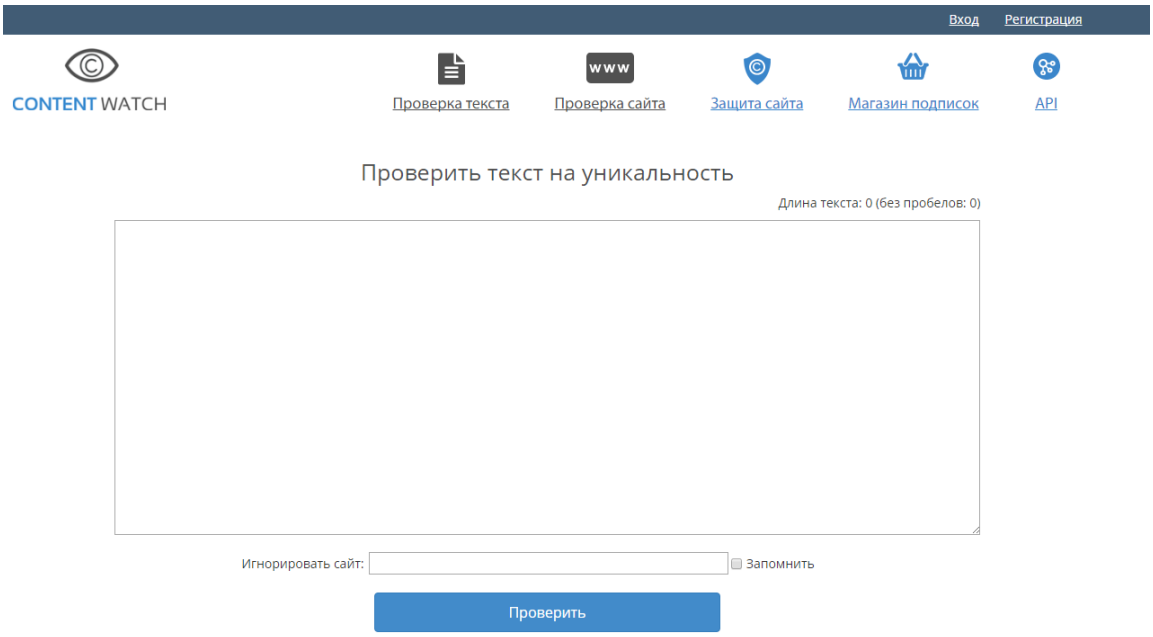

Рисунок 1.4 – Головна сторінка сайту системи «Content-watch»

При перевірці на унікальність він використовує власний алгоритм пошуку в Інтернеті сайти можуть містити матеріали повні або часткові копії заданого тексту.

Перевагою даної системи є:

– на основі запропонованих варіантів підраховується загальна унікальність тексту у відсотках;
– є можливість подивитися, які частини тексту були знайдені на кожній з проаналізованих сторінок.
– Недоліками є:
– довжина тексту до 3000 символів (розширюється до 10.000 символів після реєстрації);
– до 5 запитів в день для одного користувача без реєстрації.

У результаті проведеного аналізу існуючих систем, з множини реалізацій перевага віддається тій, яка найбільш проста у використанні та має найбільший функціонал по пошуку плагітау. Можна виділити головні якості, які необхідні для цієї системи: точність, простота, легкість у впровадженні.

2 ВИБІР І ОБГРУНТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ТА ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ЇЇ РЕАЛІЗАЦІЇ

2.1 Класифікація систем за архітектурою

Архітектура інформаційної комп'ютерної системи будується на основі апаратної частини (ЕОМ), телекомунікаційного і програмного забезпечення. Рівень розвитку кожної із складових визначений досконалістю інформаційної системи, технологією обробки даних, що зумовило виникнення таких схем обробки даних: телеобробка; файл-сервер; клієнт-сервер; Internet-система; сховище даних і система оперативної аналітичної обробки даних тощо.

2.1.1 Архітектура файл-сервер

Інформаційна система цього типу складається з трьох компонент: сервер баз даних, клієнт (персональний комп'ютер із клієнтськими застосуваннями і СУБД), мережа і комунікаційне програмне забезпечення (рис. 2.1).

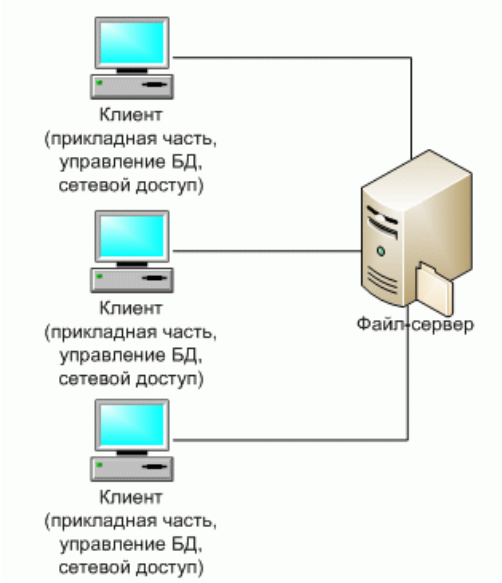

Рисунок 2.1 – Архітектура файл-сервер

На сервері розташовані СУБД і файли, які необхідні для роботи клієнтських застосувань. Клієнтські застосування і їхні персональні СУБД розташовані та функціонують на окремих робочих станціях і звертаються до файлового сервера тільки в міру потреби отримання доступу до файлів.

Сервер відбирає з бази потрібні файли (а не окремі їх записи), які мережею відправляються клієнтові для опрацювання. Таким чином, файловий сервер функціонує як сумісно використовуваний жорсткий диск. Архітектура з використанням файлового сервера характеризується такими основними недоліками: великий обсяг мережевого графіка; на кожній робочій станції має бути повна копія користувацької СУБД; управління паралельністю, відновленням і цілісністю бази даних ускладнюється, оскільки доступ до одних і тих самих файлів здійснюється одночасно кількома СУБД.

2.1.2 Архітектура клієнт-сервер

Клієнт-серверна інформаційна система складається з трьох основних компонент: програмне забезпечення сервера; програмне забезпечення кінцевого користувача; проміжне програмне забезпечення (рис. 2.2).

Рисунок 2.2 – Дворівнева архітектура клієнт-сервер

Програмне забезпечення сервера забезпечує обслуговування клієнтів. Для реалізації архітектури клієнт-сервер зазвичай використовують багатокористувацькі СУБД, наприклад, Oracle або Microsoft SQL Server. У таких СУБД передбачені механізми блокування та елементи управління багатокористувацьким доступом, які забезпечують захист даних від небезпеки паралельного доступу. Крім цього, серверу баз даних доводиться охороняти дані від несанкціонованого доступу, оптимізувати запити до бази

даних, забезпечувати цілісність даних і контроль завершення транзакцій. У клієнт-серверній організації клієнти можуть бути досить "тонкими", а сервер має бути "товстим" настільки, щоб задовольняти потреби всіх клієнтів.

До програмного забезпечення кінцевого користувача відносять засоби розробки програм і генератори звітів, у тому числі електронні таблиці і текстові процесори. За допомогою цього програмного забезпечення користувачі встановлюють зв'язок із сервером, формують запити, які автоматично генеруються в запити мовою SQL і відправляються на сервер. Сервер приймає і опрацьовує запити, а потім передає отримані результати клієнтам. Проміжне програмне забезпечення – це та частина системи "клієнт-сервер", яка пов'язує програмне забезпечення кінцевого користувача із сервером.

Схема клієнт-сервер проста: клієнт направляє серверу запит на потрібні дані; сервер їх приймає, опрацьовує і відправляє клієнтові тільки ті дані, які були замовлені. Дворівнева модель клієнт-сервер оптимальна для підприємств із кількістю користувачів меншою за 100, оскільки операційна система сервера під час обслуговування великої кількості клієнтів надто перевантажується управлінням численними підключеннями до сервера.

Трирівнева модель, на відміну від дворівневої, розв'язує проблеми масштабування. У разі використання трирівневої моделі, окрім клієнта і сервера, є ще й додатковий проміжний ланцюг (сервер застосування), який управляє транзакціями – аналізує запити, організує їх чергу, спрямовує запити на виконання тощо (рис. 2.3).

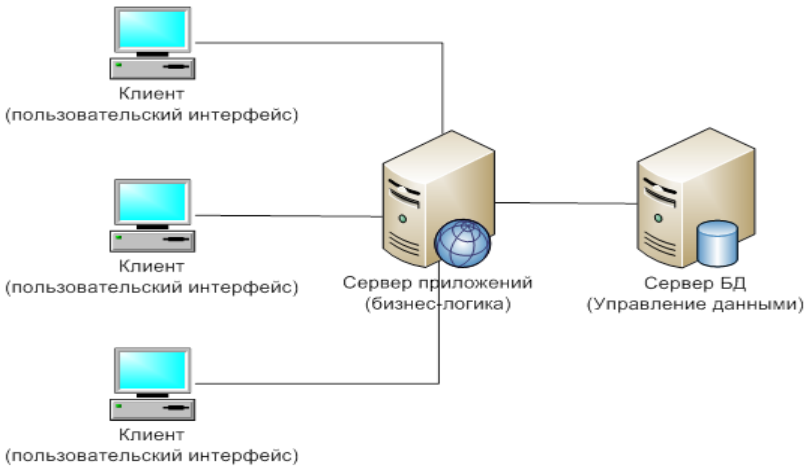

Рисунок 2.3 – Трирівнева архітектура клієнт-сервер

Клієнт-серверні інформаційні системи мають низку переваг порівняно з файл-серверними інформаційними системами. По-перше, знижується мережевий трафік при виконанні запитів. По-друге, архітектура клієнт-сервер стає незамінною, коли кількість користувачів, які одночасно користуються тими самими даними, перевищує тисячі користувачів. Ще однією перевагою архітектури клієнт-сервер є можливість збереження бізнес-правил на сервері, що дає змогу уникнути дублювання коду в різних застосуваннях, які використовують загальну базу даних. Окрім перерахованих переваг, сучасні серверні СУБД мають широкі можливості управління користувацькими привілеями і правами доступу до різноманітних об'єктів бази даних, резервного копіювання та архівації даних, а також оптимізації виконання запитів.

2.2 Вибір архітектури системи

Дана система має трирівневу архітектуру, що передбачає наявність наступних компонент програми: клієнтський додаток (зазвичай говорять «тонкий клієнт» або термінал), підключений до сервера додатків, який в свою чергу підключений до серверу бази даних.

Клієнт – це(зазвичай графічний) компонент, який представляє перший рівень, власне для кінцевого користувача. Перший рівень не повинен мати прямих зв'язків з базою даних (за вимогами безпеки), не повинен бути навантаженим основною бізнес-логікою (за вимогами масштабованості) і зберігати стан програми (за вимогами надійності). На перший рівень може бути винесена і зазвичай виноситься найпростіша бізнес-логіка: інтерфейс авторизації, алгоритми шифрування, перевірка значень, що вводяться, на допустимість і відповідність формату, нескладні операції (сортування, групування, підрахунок значень) з даними, вже завантаженими на термінал.

Сервер додатків розташовується на другому рівні. На другому рівні зосереджена більша частина бізнес-логіки. Поза ним залишаються фрагменти, що експортуються на термінали (див. вище), а також розміщені в третьому рівні збережені процедури і тригери.

У «правильної» (з точки зору безпеки, надійності, масштабування) конфігурації сервер бази даних міститься на виділеному комп'ютері (або кластері), до якого по мережі підключені один або кілька серверів додатків, до яких, в свою чергу, по мережі підключаються термінали.

У порівнянні з клієнт-серверною або файл-серверною архітектурою можна виділити такі переваги трирівневої архітектури:

- масштабованість;
- конфігурованість – ізольованість рівнів один від одного дозволяє (при правильному розгортанні архітектури) швидко і простими засобами переконфігурувати систему при виникненні збоїв або при плановому обслуговуванні на одному з рівнів;
- високий рівень безпеки;
- висока надійність;
- низькі вимоги до швидкості каналу (мережі) між терміналами і сервером додатків;
- низькі вимоги до продуктивності і технічних характеристик терміналів, як наслідок зниження їхньої вартості. Терміналом може виступати не тільки комп'ютер, але і, наприклад, мобільний телефон.

Недоліки випливають з переваг. У порівнянні с клієнт-серверною або файл-серверною архітектурою можна виділити наступні недоліки трирівневої архітектури:

- вища складність створення додатків;
- складніша у розгортанні і адмініструванні;
- високі вимоги до продуктивності серверів додатків і сервера бази даних, а, отже, і висока вартість серверного обладнання;

Незважаючи на недоліки, які в більшій мірі пов’язані з вимогами до продуктивності тих чи інших частин серверу, використовуватися буде саме трирівнева архітектура (рис. 2.4), так як переваг значно більше ніш недоліків.

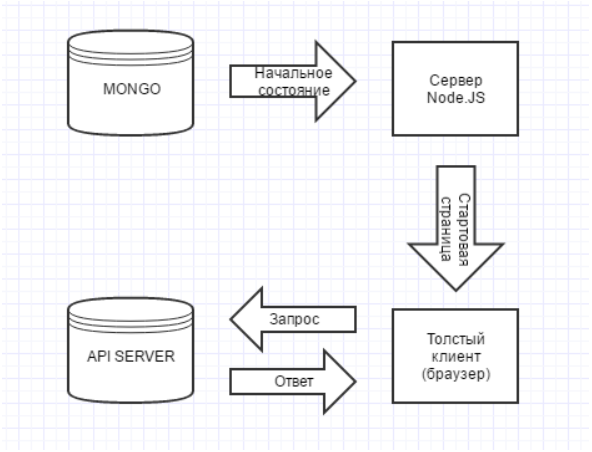

Рисунок 2.4 – Трирівнева архітектура клієнт-сервер системи виявлення плагіату

2.3 Вибір системи управління базами даних

MongoDB реалізує новий підхід до побудови баз даних, де немає таблиць, схем, запитів SQL, зовнішніх ключів і багатьох інших речей, які притаманні об'єктно-реляційних баз даних.

З часів динозаврів було звичайною справою зберігати всі дані в реляційних базах даних (MS SQL, MySQL, Oracle, PostgresSQL). При цьому було не так важливо, а чи підходять реляційні бази даних для зберігання даного типу даних чи ні.

На відміну від реляційних баз даних MongoDB пропонує документообігу орієнтовану модель даних, завдяки чому MongoDB працює швидше, має кращу масштабованість, її легше використовувати.

Але, навіть з огляду на всі недоліки традиційних баз даних і гідності MongoDB, важливо розуміти, що завдання бувають різні і методи їх вирішення бувають різні. В якійсь ситуації MongoDB дійсно поліпшить продуктивність вашої програми, наприклад, якщо треба зберігати складні за структурою дані. Іншим же ситуації краще буде використовувати традиційні реляційні бази даних. Крім того, можна використовувати змішані підхід: зберігати один тип даних в MongoDB, а інший тип даних – в традиційних БД.

Вся система MongoDB може представляти не тільки одну базу даних, що знаходиться на одному фізичному сервері. Функціональність MongoDB дозволяє розташувати кілька баз даних на декількох фізичних серверах, і ці бази даних зможуть легко обмінюватися даними і зберігати цілісність.

Одним з популярних стандартів обміну даними та їх зберігання є JSON (JavaScript Object Notation). JSON ефективно описує складні за структурою дані. Спосіб зберігання даних в MongoDB в цьому плані схожий на JSON, хоча формально JSON не використовується. Для зберігання в MongoDB застосовується формат, який називається BSON (Бісон) або скорочення від binary JSON.

BSON дозволяє працювати з даними швидше: швидше виконується пошук і обробка. Хоча треба зазначити, що BSON на відміну від зберігання даних в форматі JSON має невеликий недолік: в цілому дані в JSON-форматі займають менше місця, ніж в форматі BSON, з іншого боку, даний недолік з лишком окупається швидкістю.

MongoDB написана на C ++, тому її легко перенести на найрізноманітніші платформи. MongoDB може бути розгорнута на платформах Windows, Linux, MacOS, Solaris. Можна також завантажити

вихідний код і самому скомпілювати MongoDB, але рекомендується використовувати бібліотеки з офсайта.

Якщо реляційні бази даних зберігають рядки, то MongoDB зберігає документи. На відміну від рядків документи можуть зберігати складну за структурою інформацію. Документ можна уявити як сховище ключів і значень.

Ключ являє просту мітку, з яким асоційоване певний шматок даних.

Однак при всіх відмінностях є одна особливість, яка зближує MongoDB і реляційні бази даних. У реляційних СУБД зустрічається таке поняття як первинний ключ. Це поняття описує якийсь стовпець, який має унікальні значення. У MongoDB для кожного документа є унікальний ідентифікатор, який називається _id. І якщо явно не вказати його значення, то MongoDB автоматично згенерує для нього значення.

Кожному ключу зіставляється певне значення. Але тут також треба враховувати одну особливість: якщо в реляційних базах є чітко окреслена структура, де є поля, і якщо якесь поле не має значення, йому (в залежності від налаштувань конкретної бд) можна привласнити значення NULL. У MongoDB все інакше. Якщо якомусь ключу не приміряючи значення, то цей ключ просто опускається в документі і не вживається.

Якщо в традиційному світі SQL є таблиці, то в світі MongoDB є колекції. І якщо в реляційних БД таблиці зберігають однотипні жорстко структуровані об'єкти, то в колекції можуть містити найрізноманітніші об'єкти, що мають різну структуру і різний набір властивостей.

Система зберігання даних в MongoDB представляє набір реплік. У цьому наборі є основний вузол, а також може бути набір вторинних вузлів. Всі вторинні вузли зберігають цілісність і автоматично оновлюються разом з оновленням головного вузла. І якщо основний вузол з якихось причин виходить з ладу, то один з вторинних вузлів стає головним.

Відсутність жорсткої схеми бази даних і в зв'язку з цим потреби при щонайменшій зміні концепції зберігання даних пересоздавать цю схему значно полегшують роботу з базами даних MongoDB і подальшим їх масштабуванням. Крім того, економиться час розробників. Їм більше не треба думати про перезапуск бази даних і витрачати час на побудову складних запитів.

Однією з проблем при роботі з будь-якими системами баз даних є збереження даних великого розміру. Можна зберігати дані в файлах, використовуючи різні мови програмування. Деякі СУБД пропонують

спеціальні типи даних для зберігання бінарних даних в БД (наприклад, BLOB в MySQL).

На відміну від реляційних СУБД MongoDB дозволяє зберігати різні документи з різним набором даних, однак при цьому розмір документа обмежується 16 мб. Але MongoDB пропонує рішення – спеціальну технологію GridFS, яка дозволяє зберігати дані за розміром більше, ніж 16 мб.

Система GridFS складається з двох колекцій. У першій колекції, яка називається files, зберігаються імена файлів, а також їх метадані, наприклад, розмір. А в іншій колекції, яка називається chunks, у вигляді невеликих сегментів зберігаються дані файлів, зазвичай сегментами по 256 кб.

2.4 Вибір мови програмування та допоміжних засобів

Для розробки клієнтської частини була використана мова JavaScript. Також були використані допоміжні мови та засоби. Розглянемо їх більш детальніше.

2.4.1 Мова розмітки гіпертекстових документів HTML

HTML (англ. Hyper Text Markup Language – Мова розмітки гіпертекстових документів) – стандартна мова розмітки веб-сторінок в Інтернеті. Більшість веб-сторінок створюються за допомогою мови HTML (або XHTML). Документ HTML оброблюється браузером та відтворюється на екрані у звичному для людини вигляді.

HTML є похідною мовою від SGML, успадкувавши від неї визначення типу документу та ідеологію структурної розмітки тексту

Попри те, що HTML – штучна комп'ютерна мова, вона не є мовою програмування.

HTML разом із каскадними таблицями стилів та вбудованими скриптами – це три основні технології побудови веб-сторінок.

HTML впроваджує засоби для:

- створення структурованого документу шляхом позначення структурного складу тексту: заголовки, абзаци, списки, таблиці, цитати та інше;
- отримання інформації із Всесвітньої мережі через гіперпосилання;
- створення інтерактивних форм;
- включення зображень, звуку, відео, та інших об'єктів до тексту.

2.4.2 Каскадні таблиці стилів CSS

Каскадні таблиці стилів (англ. Cascading Style Sheets або скорочено CSS) – спеціальна мова, що використовується для опису сторінок, написаних мовами розмітки даних.

Найчастіше CSS використовують для візуальної презентації сторінок, написаних HTML та XHTML, але формат CSS може застосовуватися до інших видів XML-документів.

Специфікації CSS були створені та розвиваються Консорціумом Всесвітньої мережі.

CSS має різні рівні та профілі. Наступний рівень CSS створюється на основі попередніх, додаючи нову функціональність або розширюючи вже наявні функції. Рівні позначаються як CSS1, CSS2 та CSS3. Профілі – сукупність правил CSS одного або більше рівнів, створені для окремих типів пристроїв або інтерфейсів. Наприклад, існують профілі CSS для принтерів, мобільних пристроїв тощо.

CSS (каскадна або блочна верстка) прийшла на заміну табличній верстці веб-сторінок. Головна перевага блочної верстки – розділення змісту сторінки (даних) та їхньої візуальної презентації.

2.4.3 Мова програмування JavaScript

JavaScript (JS) – динамічна, об'єктно-орієнтована мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується як частина браузера, що надає можливість коду на стороні клієнта (такому, що виконується на пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки. Мова JavaScript також використовується для програмування на стороні серверу (подібно до таких мов програмування, як Java і C#), розробки ігор, стаціонарних та мобільних додатків, сценаріїв в прикладному ПЗ (наприклад, в програмах зі складу AdobeCreativeSuite), всередині PDF-документів тощо.

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація,

автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

Незважаючи на схожість назв, мови Java та JavaScript є двома різними мовами, що мають відмінну семантику, хоча й мають схожі риси в стандартних бібліотеках та правилах іменування. Синтаксис обох мов отриманий «у спадок» від мови C, але семантика та дизайн JavaScript є результатом впливу мов Self та Scheme.

2.4.4 Бібліотека React.js

React.js, здебільшого називають React, це open-source JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових додатків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

React дозволяє розробникам створювати великі веб-додатки, які використовують дані, які змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у додатках. Це відповідає View у шаблоні модель-вид-контролер (MVC), і може бути використаний в поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови веб-додатків.

2.4.5 Бібліотека Redux

Redux є передбачуваним контейнером стану для javascript додатків. Це дозволяє вам створювати додатки, які ведуть себе однаково в різних середовищах (клієнт, сервер і нативні додатки), а також просто тестуються. крім того, це забезпечує великий досвід налагодження, наприклад редагування коду в реальному часі в поєднанні з time traveling. Ви можете використовувати redux разом з react або з будь-якої іншої view-бібліотекою. це крихітна бібліотека (2kb, включаючи залежності). У міру того, як вимоги до односторінковим javascript додатків стають все більш високими, ми змушені керувати все більшою кількістю станів (state) за допомогою javascript. Ці стани можуть включати в себе відповіді сервера, кешированниє

дані, а також дані створені локально, але ще не збережені на сервері. Це також відноситься до ui-станів, таким як активний маршрут (route), виділений таб, показаний спиннер або нумерація сторінок і т.д.

Управляти постійно змінюються станами складно. Якщо модель може оновити іншу модель, то уявлення може оновити модель, яка оновлює іншу модель, а це, в свою чергу, може викликати оновлення іншої думки. У якийсь момент ви більше не знаєте що відбувається у вашому додатку. Ви більше не можете контролювати коли, чому, і як стан оновилося. Коли система стає непрозорою і недетермінованої, важко виявити помилки або додавати нову функціональність.

Це досить кепсько, беручи до уваги нові вимоги стають звичними для фронтенд розробки, такі як, обробка оптимістичних оновлень (optimistic updates), рендер на сервері, вилучення даних перед виконанням переходу на сторінку і так далі. Як frontend розробники, ми намагаємося впоратися зі складністю, з якою ми ніколи не мали справи перш, і тому неминуче виникає запитання: настав час здатися?

Ця складність виникає через те, що ми змішуємо дві концепції, які дуже нелегкі для розуміння: зміни (mutation) і асинхронність (asynchronicity). Я називаю їх ментос і кола. Обидві ці концепції можуть бути прекрасними окремо, але разом вони перетворюються в бардак. Бібліотеки, аналогічні react, намагаються вирішити цю проблему на рівні уявлення, видаляючи асинхронність і пряме маніпулювання dom. Проте, react залишає управління станом даних за вами. І тут в справу вступає redux.

Йдучи слідами flux, cqrs і event sourcing, redux намагається зробити зміни стану передбачуваними, шляхом введення деяких обмежень на те, як і коли можуть відбутися оновлення. Ці обмеження відображені в трьох принципах redux налагодження, наприклад редагування коду в реальному часі в поєднанні з time traveling.

Єдине джерело правди. Стан всього вашого застосування збережено в дереві об'єктів всередині одного сховища.

Це полегшує створення універсальних програм. Стан на сервері може бути серіалізіровано і відправлено на клієнт без додаткових зусиль. Це спрощує налагодження програми, коли ми маємо справу з єдиним деревом стану. Ви також можете зберігати стан вашого застосування для прискорення процесу розробки. І з єдиним деревом стану ви отримуєте функціональність типу Undo / Redo з коробки.

Стан тільки для читання. Єдиний спосіб змінити стан – це застосувати дію – об'єкт, який описує, що трапиться.

Це гарантує, що уявлення або функції, що реагують на події мережі (network callbacks), ніколи не змінять стан безпосередньо. Оскільки всі зміни централізовані і застосовуються послідовно в строгому порядку, тому немає необхідності стежити за "гонкою станів". Дії – це всього лише прості об'єкти, тому вони можуть бути залоговані, серіалізіровани, збережені і потім відтворені, для налагодження або тестування.

Мутации написаны, как чистые функции Для визначення того, як дерево стану буде трансформовано діями, ви пишете чисті редюсери. Редюсери – це просто чисті функції, які беруть попередній стан і дію і повертають новий стан. Не забувайте повертати новий об'єкт стану, замість того, щоб змінювати попереднє. Ви можете почати з одного редюсера, але в подальшому, коли ваш додаток розростеться, ви можете розділити його на більш дрібні редюсери, які керують окремими частинами дерева стану. Оскільки редюсери – це просто функції, ви можете контролювати порядок, в якому вони викликаються, відправляти додаткові дані або навіть писати переіспользуемие редюсери для загальних завдань, наприклад для пагінацію.

2.4.6 Транспайлер Babel.js

Babel.JS – це транспайлер, переписувати код на ES-2015 код на попередньому стандарті ES5.

Він складається з двох частин. Власне транспайлер, який переписує код.

Поліфілл, який додає методи Array.from, String.prototype.repeat і інші.

На сторінці можна поекспериментувати з транспайлером: зліва вводиться код в ES-2015 року, а справа з'являється результат його перетворення в ES5.

Зазвичай Babel.JS працює на сервері в складі системи збирання JS-коду (наприклад webpack або brunch) і автоматично переписує весь код в ES5.

Налаштування такої конвертації тривіальна, єдино – потрібно підняти саму систему збирання, а додати до неї Babel легко, плагіни є до будь-якої з них.

Якщо ж хочеться «погратися», то можна використовувати і браузерні варіант Babel.

2.4.7 Утиліта Webpack

Webpack – це утиліта для збірки бандлів і оптимізації модулів JavaScript та інших ресурсів для фронтенда. Якщо ви вже користувалися RequireJS або Browserify, то є всі шанси, що ви полюбите webpack так само, як і я. При разработке большого проекта очень часто возникает необходимость разбивать код на отдельные модули, которые будут взаимодействовать между собой. На сегодняшний день для этого существует два подхода: это AMT и CommonJS. Они оба позволяют разрабатывать изолированные модули, не думать о порядке их загрузки, собирать все наши модули в один js-файл, безопасно подключать сторонние библиотеки.

2.4.8 Методологія БЕМ

БЕМ (Блок, Елемент, Модифікатор) – компонентний підхід до веб-розробки. В його основі лежить принцип поділу інтерфейсу на незалежні блоки. Він дозволяє легко і швидко розробляти інтерфейси будь-якої складності і повторно використовувати існуючий код, уникаючи «Copy-Paste».

Блок – це функціонально незалежний компонент сторінки, який може бути повторно використаний. В HTML блоки представлені атрибутом class.

Блоки можуть бути вкладені в будь-які інші блоки. Наприклад, блок head може включати логотип (logo), форму пошуку (search) і блок авторизації (рис. 2.5).

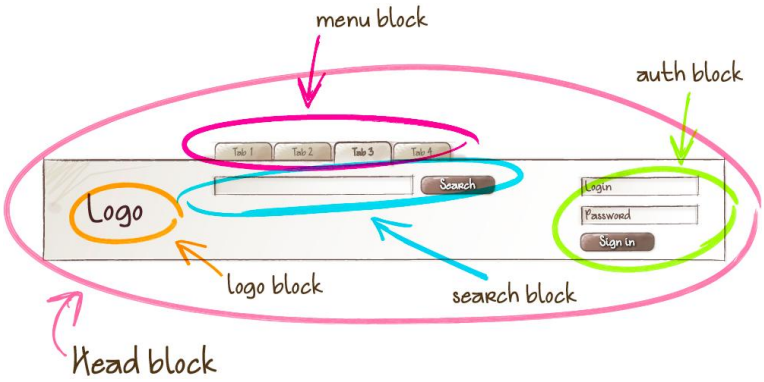

Рисунок 2.5 – Структура в БЕМ

Блоки можна переміщати в межах однієї сторінки, різних сторінок або проектів. Незалежна реалізація блоку дозволяє змінювати його положення на сторінці і забезпечує коректну роботу і зовнішній вигляд.

Так, наприклад, логотип і форму авторизації можна поміняти місцями. При цьому вносити зміни в CSS або JavaScript-код блоку не потрібно (рис. 2.6).

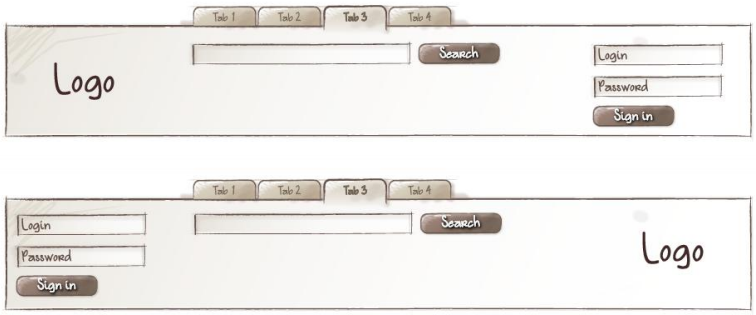

Рисунок 2.6 – Реструктуризація в методології БЕМ

В інтерфейсі може одночасно бути присутнім кілька примірників одного і того ж блоку (рис 2.7).

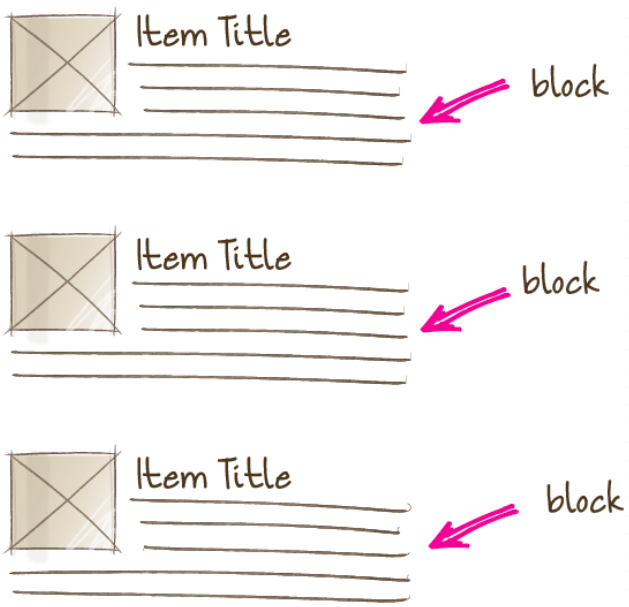

Рисунок 2.7 – Кілька примірників блоку в методології БЕМ

Елемент – це складова частина блоку, яка не може використовуватися у відриві від нього.

Наприклад, пункт меню поза контекстом блоку меню не використовується, значить є елементом (рис 2.8).

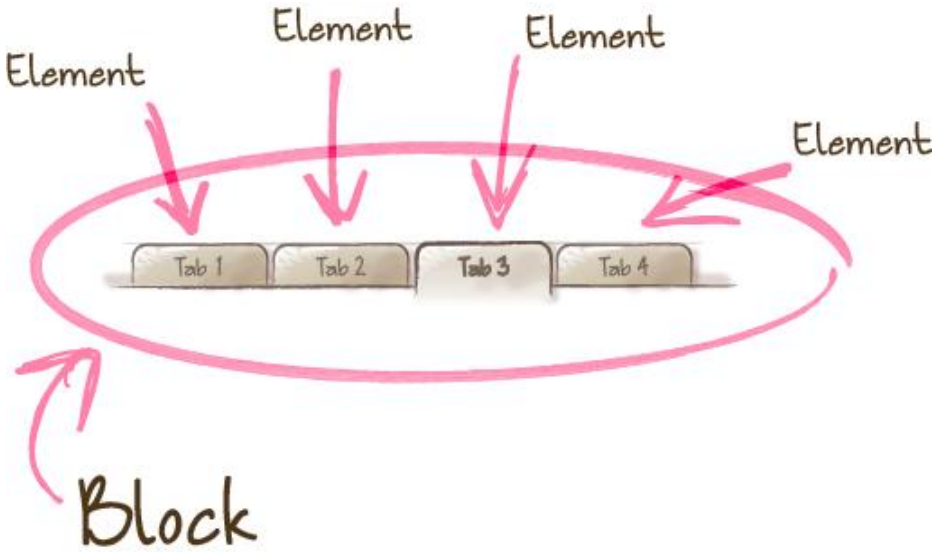

Рисунок 2.8 – Елементи в методології БЕМ

Модифікатор – це БЕМ-сутність, яка визначає зовнішній вигляд, стан і поведінку блоку або елементу.

Використання модифікаторів опціонально. За своєю суттю модифікатори схожі на атрибути в HTML. Один і той же блок виглядає по-різному завдяки застосуванню модифікатора. Наприклад, зовнішній вигляд блоку меню (menu) може змінюватися в залежності від застосованого модифікатора (рис 2.9).

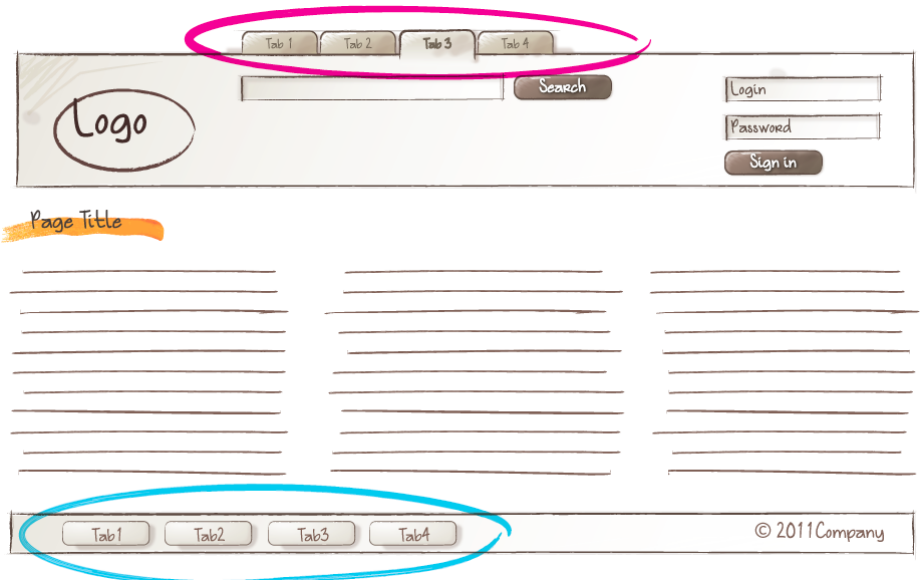

Рисунок 2.9 – Модифікатори в методології БЕМ

Модифікатори можуть змінюватися як в процесі роботи блоку (наприклад, як реакція на DOM-події блоку), так і за запитом з інших блоків.

Наприклад, при кліці по кнопці Sign In (DOM-подія click), в разі невірно заповнених полів Login або Password, на прихований блок повідомлень про помилки встановити модифікатор (visible).

Перевизначення блоку – це зміна реалізації блоку шляхом додавання йому нових особливостей на іншому уровне.ий блок повідомлень про помилки встановити модифікатор (visible).

Кінцева реалізація блоку може бути розділена за різними рівнями перевизначення. Кожен наступний рівень додає або перекриває вихідну реалізацію блоку. Кінцевий результат збирається з окремих технологій реалізації блоку з усіх рівнів перевизначення послідовно в заданому порядку (рис 2.10).

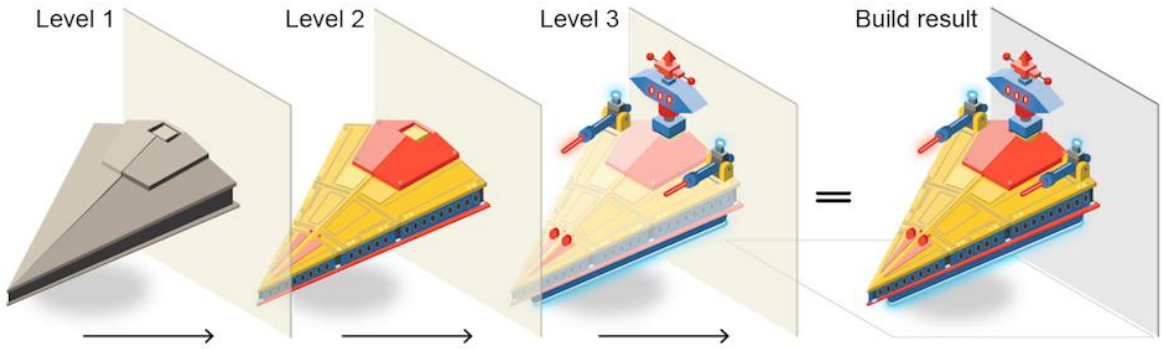

Рисунок 2.10 – Перевизначення блоків в методології БЕМ

2.4.9 Скриптова метамова Sass

Sass (Syntactically Awesome Stylesheets) – це скриптова метамова, який компілюється в звичайні CSS-стилі. Якщо ви добре знайомі з CSS + HTML, то з SASS розберетеся за кілька днів. Всі, хто стикається з CSS розміром більше 500 рядків, мають справу з головним болем на тему того, як би його спростити. На жаль, з часів розробки стандартів каскадних стилів їх структура кардинально не змінювалася. Вимоги до верстки, кому я буду брехати, – ускладнилися в рази. Якщо колись 50-70 рядків стилів могли оформити простий сайт, то сьогодні такого обсягу вистачить хіба що на header.

Розширення SASS-файлів можуть бути .sass і .scss – це залежить від обраного синтаксису. Браузер, втім, не розуміє жодного з них, тому для взаєморозуміння потрібно використовувати компілятор. Його завдання – привести SASS в зрозумілий класичний CSS, який буде розпізнано будь-яким браузером. Навіть шостим Експлорером на кінній тязі.

Роль компілятора може виконувати серверний js або програма, встановлена у вас на робочій машині і моніторять зміни в робочих файлах.

У мови є два основних "накреслення": SASS і новіший SCSS. Відмінності між ними невеликі, проте порушення правил синтаксису не дозволить скомпілювати файл. У SASS-синтаксисі немає фігурних дужок, вкладеність елементів в ньому реалізована за допомогою відступів, а стильові правила обов'язково відокремлені новими рядками.

Незалежно від синтаксису, SCSS назад сумісний з CSS. Тобто будь-який CSS обов'язково буде дійсним SCSS-кодом.

Спочатку розкажу в двох словах і далі трохи докладніше. Мені в CSS завжди не вистачало змінних і заважали хакі для кросбраузерності. Нижче на прикладах ви побачите, що SASS вирішує ці дві проблеми блискуче.

Sass дозволяє призначати змінні – і це одна з ключових переваг. Змінна, за аналогією з php, починається зі знака долара ($), значення присвоюються за допомогою двокрапки. Змінні в Sass можна розділити на 4 типи:

- число (int);
- рядок (string);
- логічний тип (так / ні, boolean);
- кольору (ім'я, імена).

2.5 Вибір серверу

В якості серверу, у односторінкових додатках, досить часто використовують NodeJS.

Node або Node.js – програмна платформа, заснована на движку V8 (здійснює трансляцію JavaScript в машинний код), що перетворює JavaScript з вузькоспеціалізованого мови в мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями введення-виведення через свій API (написаний на C ++), підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з JavaScript-коду. Node.js застосовується переважно на сервері, виконуючи роль веб-сервера, але є можливість розробляти на Node.js і десктопні віконні додатки (за допомогою NW.js, AppJS або Electron для Linux, Windows і Mac OS) і навіть

програмувати мікроконтролери (наприклад, tessel і espruino). В основі Node.js лежить подієво-орієнтоване і асинхронне (або реактивне) програмування з неблокуючим в / в.

Node.js добре поводить себе в додатках реального часу, так як задіє push-технологію через веб-сокети. Ну, як вже говорилося, через 20 років використання вищезгаданої парадигми з'явилися такі двонаправлені додатки, де зв'язок може ініціювати як клієнт, так і сервер, а потім приступати до вільного обміну даними. Така технологія різко контрастує з типовою парадигмою веб-відгуків, де комунікацію завжди ініціює клієнт. Крім того, вся технологія заснована на відкритому веб-стеку (HTML, CSS і JS), робота йде через стандартний порт 80.

Багато хто заперечуэ, що все це було у нас вже не один рік – у вигляді Flash і Java-аплетів – але насправді це були просто пісочниці, які брали Веб в якості транспортного протоколу для доставки даних клієнта. Крім того, вони працювали ізольовано і часто діяли через нестандартні порти, що могло вимагати додаткових прав доступу і т.п.

Node.js при всіх його достоїнствах в даний час грає ключову роль в технологічному стеку багатьох видатних компаній, що безпосередньо залежать від унікальних властивостей Node.

Основна ідея Node.js: використовувати неблокірующій подієво-орієнтований введення / виведення, щоб залишатися легковажним і ефективним при поводженні з додатками, що обробляють великі обсяги даних в реальному часі і функціонуючими на розподілених пристроях.

По суті, це означає, що Node.js не є платформою на всі випадки життя, яка буде домінувати в світі веб-розробки. Навпаки, це платформа для вирішення строго визначених завдань. Розуміти це абсолютно необхідно. Зрозуміло, не варто використовувати Node.js для операцій, інтенсивно навантажують процесор, більше того – застосування Node.js в важких обчисленнях фактично анулює всі його переваги. Node.js дійсно хороший для створення швидких масштабованих мережевих додатків, оскільки дозволяє одночасно обробляти величезну кількість з'єднань з великою пропускною здатністю, що рівноцінно високою масштабованості.

Тонкощі роботи Node.js «під капотом» досить цікаві. У порівнянні з традиційними варіантами веб-сервісів, де кожне з'єднання (запит) породжує новий потік, навантажуючи оперативну пам'ять системи і, врешті-решт, розбираючи цю пам'ять без залишку, Node.js набагато економічніше: він працює в єдиному потоці (рис. 2.11), при викликах використовує

неблокуючий введення / виведення, дозволяє підтримувати десятки тисяч конкурентних з'єднань (які існують в циклі подій).

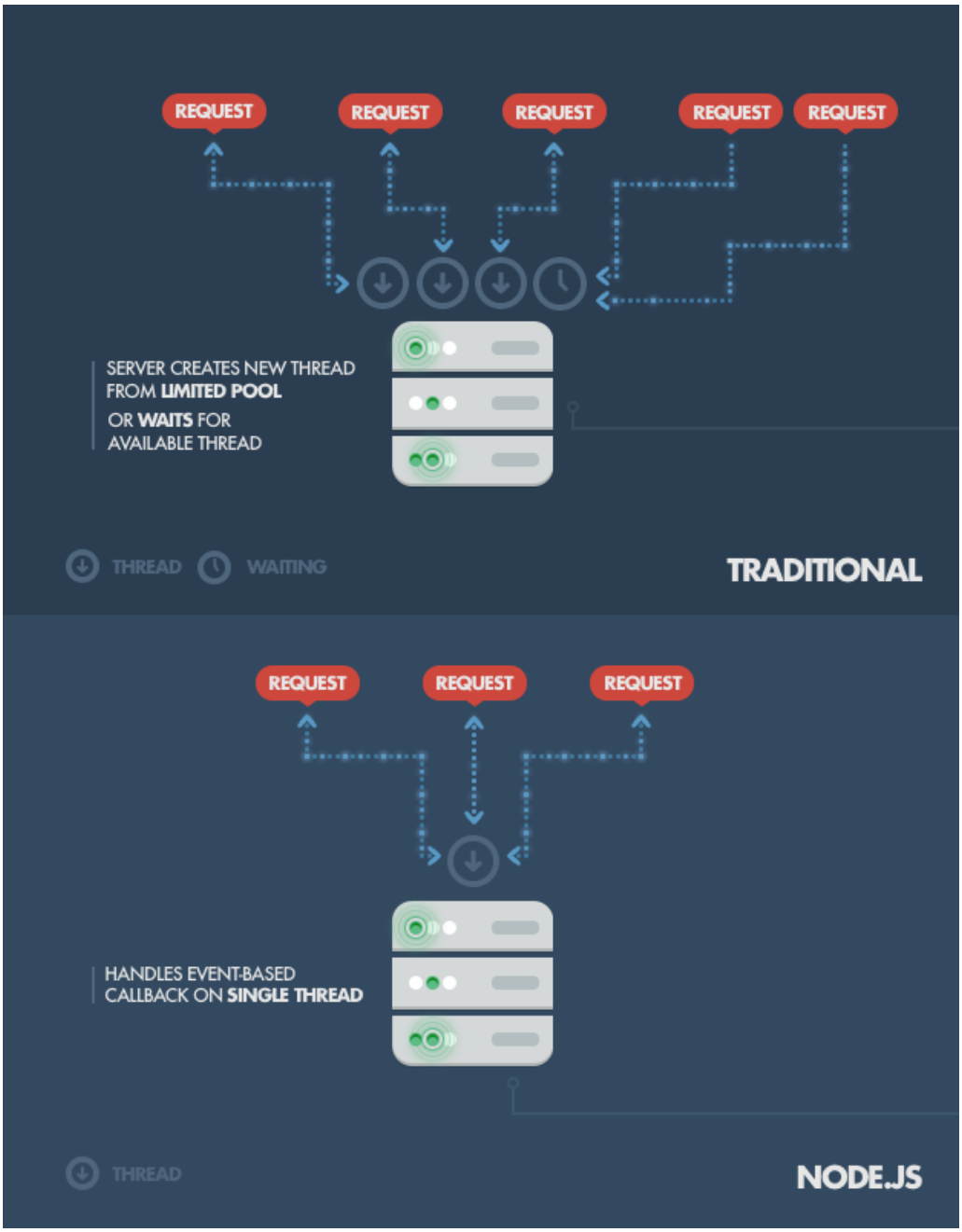

Рисунок 2.11 – Схема роботи NodeJS

Простий розрахунок: припустимо, кожен потік потенційно може зажадати 2 Мб пам'яті і працює в системі з 8 Гб оперативної пам'яті. В такому випадку ми теоретично можемо розраховувати максимум на 4000 конкурентних з'єднань плюс витрати на перемикання контексту між потоками.

Саме з таким сценарієм доводиться мати справу при використанні традиційних веб-сервісів. Node.js, уникаючи всього цього, може масштабуватися більш ніж до мільйона конкурентних з'єднань (як експеримент для підтвердження концепції).

Зрозуміло, виникає питання про поділ єдиного потоку між усіма клієнтськими запитами, саме в цьому полягає основна «пастка» при написанні додатків із застосуванням Node.js. По-перше, складні обчислення можуть забити єдиний потік Node.js, що може спричинити проблеми для всіх клієнтів (докладніше про це нижче), оскільки вхідні запити будуть блокуватися аж до завершення запитаного обчислення. По-друге, розробники повинні бути дуже уважні і не допускати спливання винятків до базового (самого верхнього) циклу подій Node.js, оскільки в іншому випадку екземпляр Node.js завершиться (фактично ж, обрушиться вся програма).

Щоб уникнути спливання винятків до самої поверхні застосовується наступний прийом: помилки передаються назад викликає стороні як параметри зворотного виклику (а не викидаються, як в інших середовищах). На випадок, якщо якесь необроблене виняток проскочить і спливе, існує безліч парадигм і інструментів, що дозволяють стежити за процесом Node і виконувати необхідне відновлення аварійно завершився примірника (хоча, призначені для користувача сеанси при цьому відновити не вдасться). Найбільш поширеними з них є модуль Forever, або робота із застосуванням зовнішніх системних інструментів upstart і monit.

2.5.1 NPM. Менеджер пакетів Node

Обговорюючи Node.js, просто необхідно згадати існуючу в ньому вбудовану підтримку управління пакетами, для якої застосовується інструмент NPM, за замовчуванням присутній в будь-яку установку Node.js. Ідея модулів NPM багато в чому схожа з Ruby Gems: це набір загальнодоступних компонентів для багаторазового використання, які легко встановити через онлайновий репозиторій; для них підтримується управління версіями і залежностями.

Повний список упакованих модулів знаходиться на сайті NPM npmjs.org, а також доступний за допомогою інструменту NPM CLI, який автоматично встановлюється разом з Node.js. Екосистема модулів абсолютно відкрита, будь-хто може опублікувати в ній власний модуль, який з'явиться в

списку сховища NPM. Короткий вступ в NPM (трохи застарий, але як і раніше актуальне) знаходиться на сайті howtonode.org/introduction-to-npm.

Деякі з найбільш популярних сучасних NPM-модулів:

- express.js, фреймворк для веб-розробки для Node.js, написаний в дусі Sinatra, де-факто – стандартний для більшості існуючих сьогодні додатків Node.js;
- connect: Connect – це розширюваний фреймворк, який працює з
- node.js як HTTP-сервера, що надає колекцію високопродуктивних «плагінів», відомих під загальною назвою «сполучна ПО» (middleware); служить основою для Express;
- socket.io і sockjs – серверна частина двох найбільш поширених сьогодні веб-сокетних компонентів;
- Jade – один з популярних шаблонизатор, написаний в дусі HAML, за замовчуванням використовується в Express.js;
- mongo і mongojs – обгортки MongoDB, що надають API для об'єктних баз даних MongoDB в Node.js;
- redis – клієнтська бібліотека Redis;
- coffee-script – компілятор CoffeeScript, що дозволяє розробникам писати програми з Node.js за допомогою Coffee;
- underscore (lodash, lazy) – Найпопулярніша допоміжна бібліотека JavaScript, упакована для використання з Node.js, а також дві аналогічні їй бібліотеки, що забезпечують підвищену продуктивність, оскільки реалізовані трохи інакше;
- forever – Ймовірно, найпоширеніша утиліта, яка забезпечує безперебійне виконання сценарію на заданому вузлі. Підтримує працездатність вашого процесу Node.js при будь-яких несподіваних збоях;

Ось що відбувається (рис. 2.12), коли один з клієнтів відправляє повідомлення:

- браузер підхоплює клацання по кнопці 'Send' за допомогою JavaScript-обробника, забирає значення з поля введення (т.е., текст повідомлення) і видає повідомлення веб-сокета, скориставшись клієнтом веб-сокетів, підключеним до нашого сервера (цей клієнт инициализируется разом з веб-сторінкою);
- серверний компонент веб-сокетного з'єднання отримує повідомлення і перенаправляє його всім іншим підключеним клієнтам широкомовною методом;
- всі клієнти отримують нове повідомлення за принципом push за допомогою веб-сокетного компонента, що виконується на веб-

сторінці. Потім вони підхоплюють контент повідомлення і оновлюють веб-сторінку, додаючи на форум новий запис;

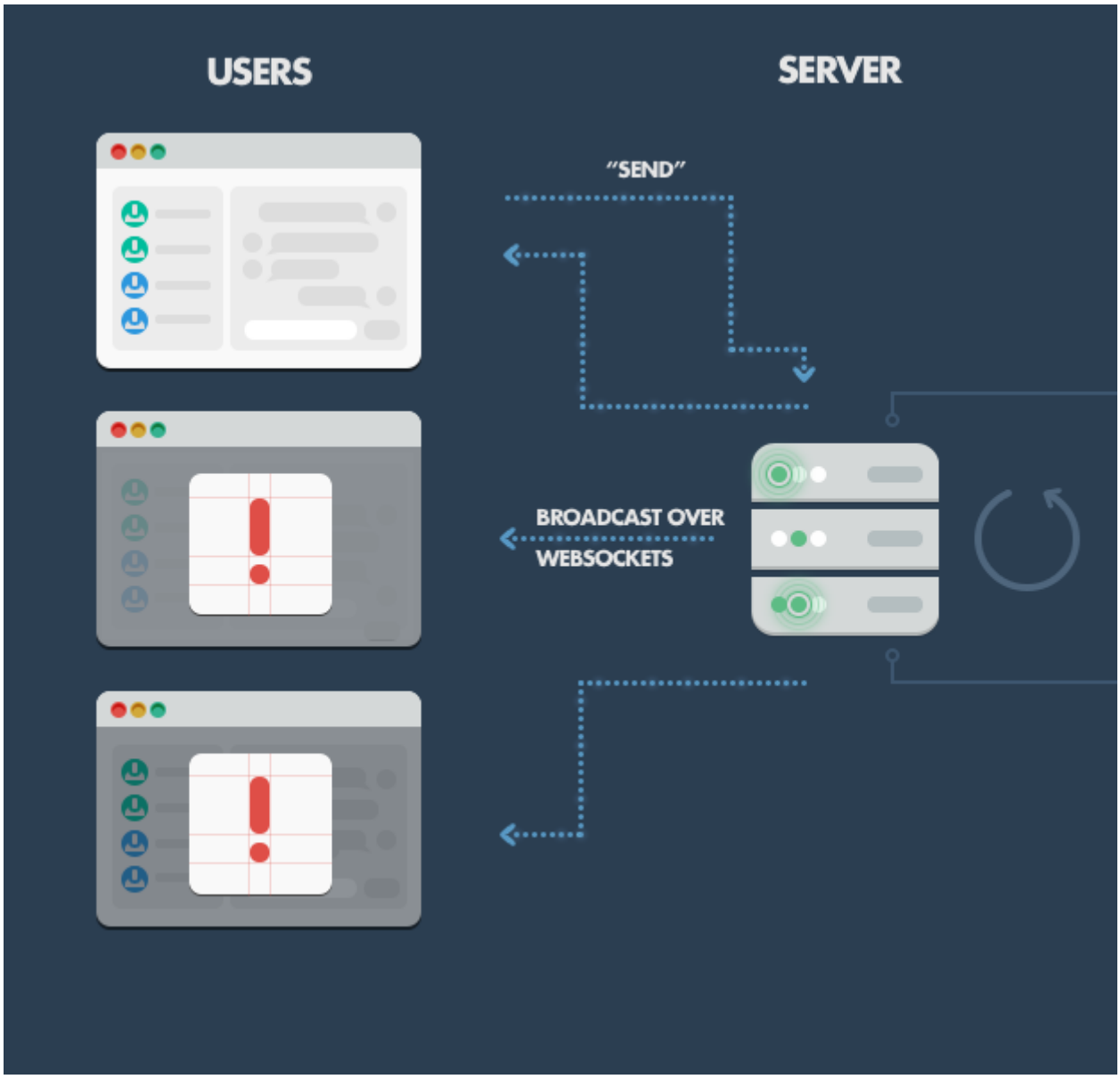

Рисунок 2.12 – Схема відправки повідомлень в NodeJS

Це найпростіший приклад. Для більш надійного рішення можна використовувати простий кеш, заснований на сховище Redis. Ще більш просунуте рішення – черга повідомлень, що дозволяє обробляти маршрутизацію повідомлень до клієнтів і забезпечує більш надійний механізм доставки, який дозволяє компенсувати тимчасові обриви з'єднання або зберігати повідомлення для зареєстрованих клієнтів, поки вони знаходяться в онлайні. Але незалежно від того, які оптимізації ви виконуєте, Node.js все одно буде діяти відповідно до тих же базовими принципами:

реагувати на події, обробляти безліч конкурентних з'єднань, підтримувати плавність користувальницьких взаємодій.

Хоча Node.js особливо хороший в контексті додатків, що працюють в реальному часі, він цілком підходить і для надання інформації з об'єктних баз даних (напр. MongoDB). Дані, збережені у форматі JSON, дозволяють Node.js функціонувати без втрати відповідності та без перетворення даних.

Наприклад, якщо ви використовуєте Rails, то вам довелося б перетворювати JSON в двійкові моделі, а потім знову надавати їх у вигляді JSON по HTTP, коли дані будуть споживатися Backbone.js, Angular.js або навіть звичайними викликами jQuery AJAX. Працюючи з Node.js, можна просто надавати ваші об'єкти JSON клієнту через REST API, щоб клієнт їх споживав. Крім того, не доводиться турбуватися про перетворення між JSON і чим завгодно ще при зчитуванні бази даних і запису в неї (якщо ви використовуєте MongoDB). Отже, ви обходитесь без безлічі перетворень, використовуючи універсальний формат серіалізації даних, застосовуваний і на клієнті, і на сервері, і в базі даних.

2.5.2 Черги вводу

Якщо ви отримуєте великі обсяги конкурентних даних, то база даних може стати вузьким місцем. Як показано вище, Node.js з легкістю обробляє конкурентні з'єднання як такі. Але оскільки звернення до бази даних є блокує операцією (в даному випадку), у нас виникають проблеми. Рішення в тому, щоб зафіксувати поведінку клієнта перед тим, як дані насправді будуть записані в базу.

При використанні такого підходу чуйність системи зберігається і під високим навантаженням, що особливо корисно, якщо клієнтові не потрібно підтвердження про те, що запис даних пройшла успішно. Типові приклади: логирование або запис даних про користувача активності (user tracking), оброблюваних по пакетному принципом і не використовуваних згодом; операції, підсумок яких повинен відображатися миттєво (наприклад, оновлення кількості «лайків» у Facebook), де прийнятна узгодженість в кінцевому рахунку, так часто використовувана в світі NoSQL.

Дані шикуються в чергу за допомогою спеціальної інфраструктури для кешування і роботи з чергами повідомлень (напр., RabbitMQ, ZeroMQ) і перетравлюються окремим процесом бази даних, призначеним для пакетного запису, або спеціальними сервісами інтерфейсу бази даних, розрахованих на

інтенсивні обчислення. Аналогічна поведінка можна реалізувати і за допомогою інших мов / фреймворків, але на іншому апаратному забезпеченні і не з такою високою і стабільною пропускною спроможністю (рис. 2.13).

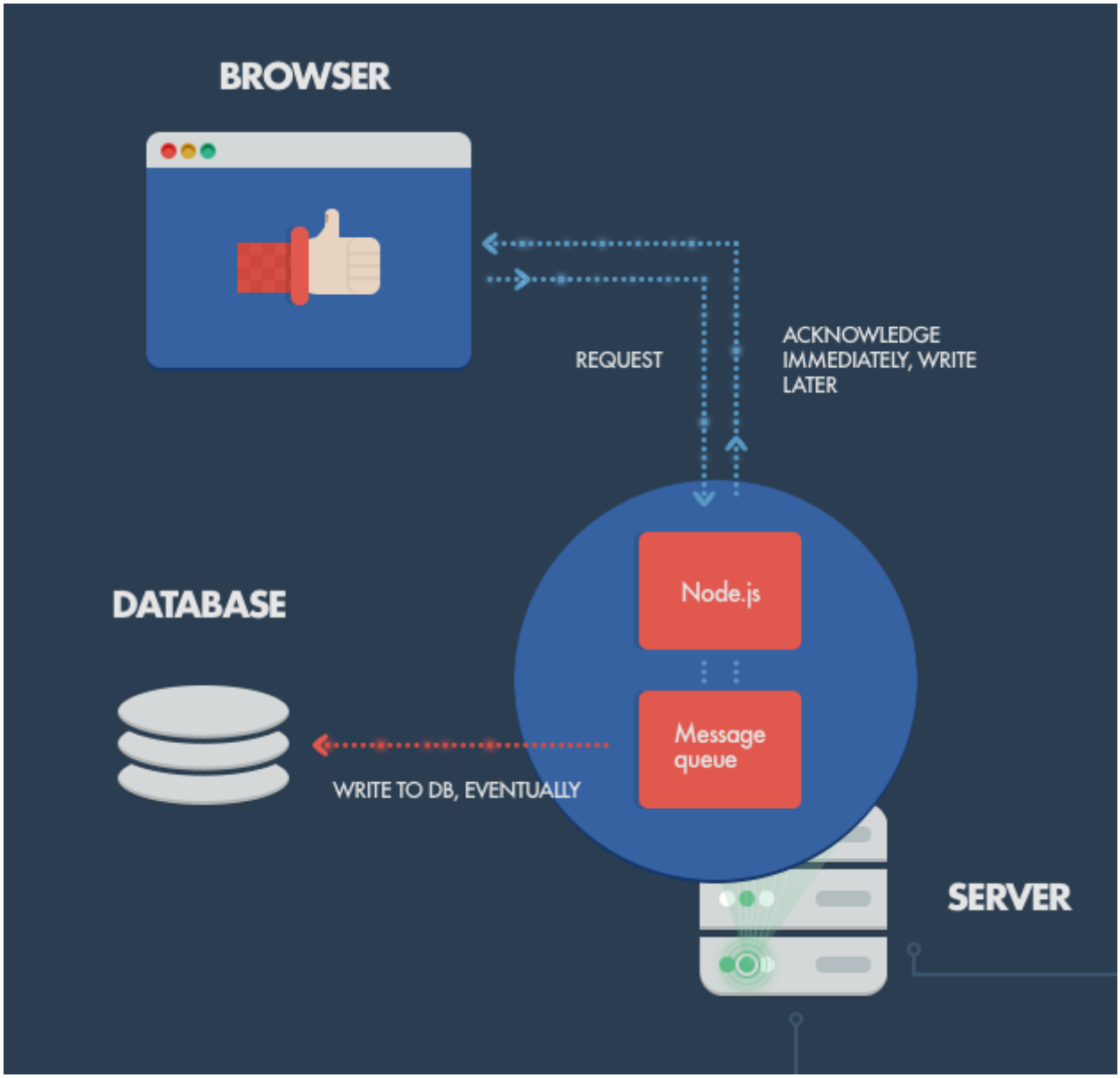

Рисунок 2.13 – Схема черг вводу в NodeJS

На більш традиційних веб-платформах HTTP-запити і відгуки трактуються як ізольовані події; але фактично вони являють собою потоки. Цим моментом можна скористатися в Node.js для створення деяких класних можливостей. Наприклад, можна обробляти файли, поки вони ще закачуються, оскільки дані надходять потоком, і ми можемо працювати з ними в онлайновому режимі. Це можна зробити, наприклад, при кодуванні

відео або аудіо в реальному часі, а також при установці проксі між різними джерелами даних (докладніше див. У наступному розділі).

Node.js цілком можна використовувати в якості серверного проксі, і в такому випадку він може обробляти велику кількість одночасних з'єднань в неблокуючому режимі. Це особливо зручно при посередництві між різними сервісами, у яких відрізняється час відгуку, або при зборі даних з багатьох джерел.

Для прикладу давайте розглянемо серверний додаток, обмінюватися інформацією зі сторонніми ресурсами, що збирає інформацію з різних джерел або зберігає такі ресурси, як зображення і відео, які потім надаються стороннім хмарних сервісів.

Хоча й існують виділені проксі-сервери, замість них зручно використовувати Node, особливо якщо проксі-інфраструктури не існує, або якщо вам потрібно рішення для локальної розробки. Тут я маю на увазі, що можна створити клієнтську програму, де буде застосовуватися сервер розробки Node.js, де ми будемо зберігати ресурси і робити проксі / заглушки для запитів до API, а в реальних умовах такі взаємодії вже будуть виконуватися за допомогою виділеного проксі сервісу (nginx, HAProxy, тощо).

Тепер давайте поговоримо про інфраструктурні аспектах. Припустимо, є SaaS-провайдер, який бажає запропонувати користувачам сторінку для відстеження сервісів (скажімо, статусну сторінку GitHub). Маючи цикл подій Node.js, можна створити потужний веб-інтерфейс, де стану сервісів будуть асинхронно перевірятися в реальному часі, а дані будуть відправлятися клієнту через веб-сокети.

Така технологія дозволяє повідомляти про статусах як внутрішніх (внутрішньокорпоративних), так і загальнодоступних сервісів в реальному часі. Давайте трохи розвинемо цю ідею і спробуємо уявити мережевий операційний центр (NOC), що відслідковує роботу додатків оператора зв'язку, провайдера хмарних сервісів / хостинг-провайдера або який-небудь фінансової організації. Все це працює у відкритому веб-стек на основі Node.js і веб-сокетів, а не Java і / або Java-аплетів.

Коли мова заходить про складні обчисленнях, Node.js залишає бажати кращого. Зрозуміло, ви не збираєтеся програмувати на Node сервер для обчислень Фібоначчі. В принципі, будь-які обчислювальні операції, сильно навантажують процесор, девальвують виграш в пропускній здатності, який в Node досягається завдяки подієво-орієнтованого неблокірующіх вводу /

виводу. Справа в тому, що будь-які вхідні запити будуть блокуватися, поки єдиний потік зайнятий переварюванням чисел.

Як було зазначено вище, Node.js однопотоковий і використовує всього одне ядро процесора. Може знадобитися реалізувати конкурентність на багатоядерному сервері, для цього команда розробників ядра Node вже займається підготовкою спеціального кластерного модуля. Крім того, ви без зусиль могли б запустити кілька екземплярів сервера Node.js за зворотним проксі з використанням nginx.

При кластеризації зберігається можливість вивантажити всі складні обчислення в фонові процеси, які працюють в більш підходящої середовищі, і забезпечити комунікацію між ними через сервер черги повідомлень, наприклад, RabbitMQ.

Хоча спочатку ваша фонова обробка могла протікати все на тому ж сервері, такий підхід забезпечує дуже високу масштабованість. Подібні служби фонової обробки легко можна розподілити на окремі робочі сервери без необхідності конфігурувати навантаження «фронтальних» веб-серверів.

Зрозуміло, такий підхід доречний і на інших платформах, але у випадку з Node.js купується та сама величезна пропускна здатність, про яку ми говорили вище, оскільки кожен запит – це маленьке завдання, що обробляється дуже швидко і ефективно.

3 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Проектування інформаційної системи за допомогою методології функціонального моделювання SADT(Стандарт IDEF0)

При проектуванні Системи виявлення плагіату, була обрана методологія SADT.

SADT (акронім від англ. Structured Analysis and Design Technique) – методологія структурного аналізу і проектування, інтегруюча процес моделювання, управління конфігурацією проекту, використання додаткових мовних засобів і керівництво проектом зі своєю графічною мовою. Процес моделювання може бути розділений на декілька етапів: опитування експертів, створення діаграм і моделей, поширення документації, оцінка адекватності моделей і прийняття їх для подальшого використання. Цей процес добре налагоджений, тому що при розробці проекту фахівці виконують конкретні обов'язки, а бібліотекар забезпечує своєчасний обмін інформацією.

SADT виникла наприкінці 60-х років в ході революції, викликаної структурним програмуванням. Коли більшість фахівців билося над створенням програмного забезпечення, мало хто намагався вирішити більш складну задачу створення великомасштабних систем, що включають як людей і машини, так і програмне забезпечення, аналогічних системам, застосовуваним в телефонному зв'язку, промисловості, управлінні та контролі за озброєнням. У той час фахівці, традиційно займалися створенням великомасштабних систем, стали усвідомлювати необхідність більшої впорядкованості. Таким чином, розробники вирішили формалізувати процес створення системи, розбивши його на наступні фази:

- аналіз – визначення того, що система буде робити;
- проектування – визначення підсистем та їх взаємодії;
- реалізація – розробка підсистем окремо, об'єднання-з'єднання підсистем в єдине ціле;
- тестування – перевірка роботи системи;
- установка – введення системи в дію;
- експлуатація – використання системи.

За допомогою графічної мови IDEF0, інформаційно-пошукова система фармацевтичних фірм постає у вигляді набору взаємопов'язаних

функціональних блоків. Моделювання засобами IDEF0, як правило, є першим етапом вивчення системи.

IDEF0-моделі складаються з трьох типів документів: графічних діаграм, тексту і глосарію. Ці документи мають перехресні посилання один на одного. Графічна діаграма – головний компонент IDEF0-моделі, що містить блоки, стрілки, з'єднання блоків і стрілок та асоційовані з ними відносини. Блоки представляють основні функції об’єкта що моделюється. Ці функції можуть бути розбиті (декомпозиція) на складові частини та представлені у вигляді більш докладних діаграм; процес декомпозиції продовжується до тих пір, поки об'єкт не буде описаний на рівні деталізації, необхідному для досягнення цілей конкретного проекту. Діаграми верхнього рівня забезпечує найбільш загальне або абстрактне описання об'єкта моделювання. За цією діаграмою слідує серія дочірніх діаграм, що дають більш детальне уявлення про об'єкт.

Кожна модель повинна мати контекстну діаграму верхнього рівня, на якій об'єкт моделювання представлений єдиним блоком з граничними стрілками. Ця діаграма називається А-0 (А мінус нуль). Стрілки на цій діаграмі відображають зв'язок об'єкта моделювання з навколишнім середовищем. Оскільки єдиний блок представляє весь об'єкт, його ім'я загальне для всього проекту. Це ж справедливо і для всіх стрілок діаграми, оскільки вони представляють повний комплект зовнішніх інтерфейсів об'єктивним та. Діаграма А-0 встановлює область моделювання та її межу.

Контекстна діаграма Системи по виявленню плагіату представлена на рис. 3.1.

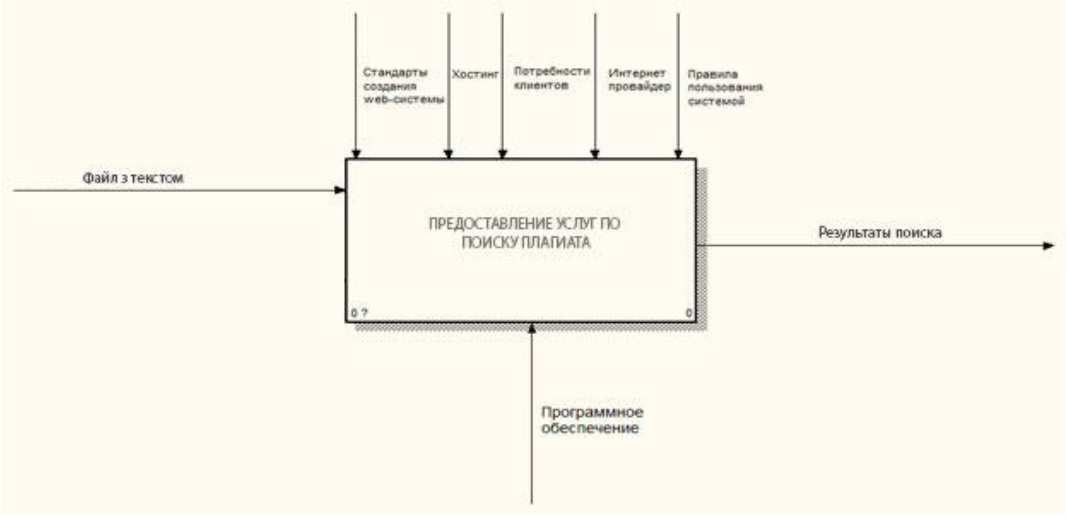

Рисунок – 3.1 Контекстна діаграма системи по виявленню плагіату

На контекстній діаграмі відображена головна робота «Системи виявлення плагіату». На вхід подається файл з текстом. Головна робота керується: правилами створення Web-системи, бажаннями замовника, особливостями хостингу, можливостями Інтернет провайдера і правилами оформлення замовлень. Виходом є: результати пошуку.

Після опису системи в цілому проводиться розбиття її на великі фрагменти. Цей процес називається функціональна декомпозиція, а діаграми, які описують кожен фрагмент і взаємодію фрагментів, називаються діаграмами декомпозиції .

Після декомпозиції контекстної діаграми проводиться декомпозиція кожного великого фрагмента системи на більш дрібні і так далі, до досягнення потрібного рівня подробиці опису. Після кожного сеансу декомпозиції проводяться сеанси експертизи – експерти предметної області вказують на відповідність реальних процесів створеним діаграмам. Знайдені невідповідності виправляються, і тільки після проходження експертизи без зауважень можна приступати до наступного сеансу декомпозиції. Так досягається відповідність моделі реальним процесам на кожному рівні декомпозиції моделі. Синтаксис опису системи в цілому і кожного її фрагмента однаковий у всій моделі.

Після декомпозиції контекстної діаграми отримуємо 3 блоку – роботи. Ці блоки представляють основні під функції початкової функції.

Функція «Розробка Web-системи» включає в себе повну розробку інформаційної системи на локальному комп'ютері. Включає в себе розробку інтерфейсу, скриптів і баз даних. Входом у неї є дані про товари, так як вони потрібні для наповнення БД. Управляється за допомогою правил створення Web-системи та правил оформлення замовлення. Механізмом є програмне забезпечення, яке потрібне для розробки. І результатом роботи є готова Web-система.

Функція «Розміщення Web-системи» служить для можливості отримати доменне ім'я, а потім помістити Web-систему на хостинг. Входом для роботи є готова Web-система для розміщення. Управляється робота бажаннями замовника, особливостями хостингу та Інтернет провайдером. Механізмом є – програмне забезпечення.

Функція «Надання послуг» служить для отримання можливості укладення угод, або, за бажанням, отримання інших доступних клієнтам послуг. Входом цієї функції є працездатна система, розміщена у мережі.

Управляється за допомогою Інтернет провайдеру та потребами клієнтів. Механізмом є – програмне забезпечення.

Діаграма декомпозиції наведена на рис. 3.2.

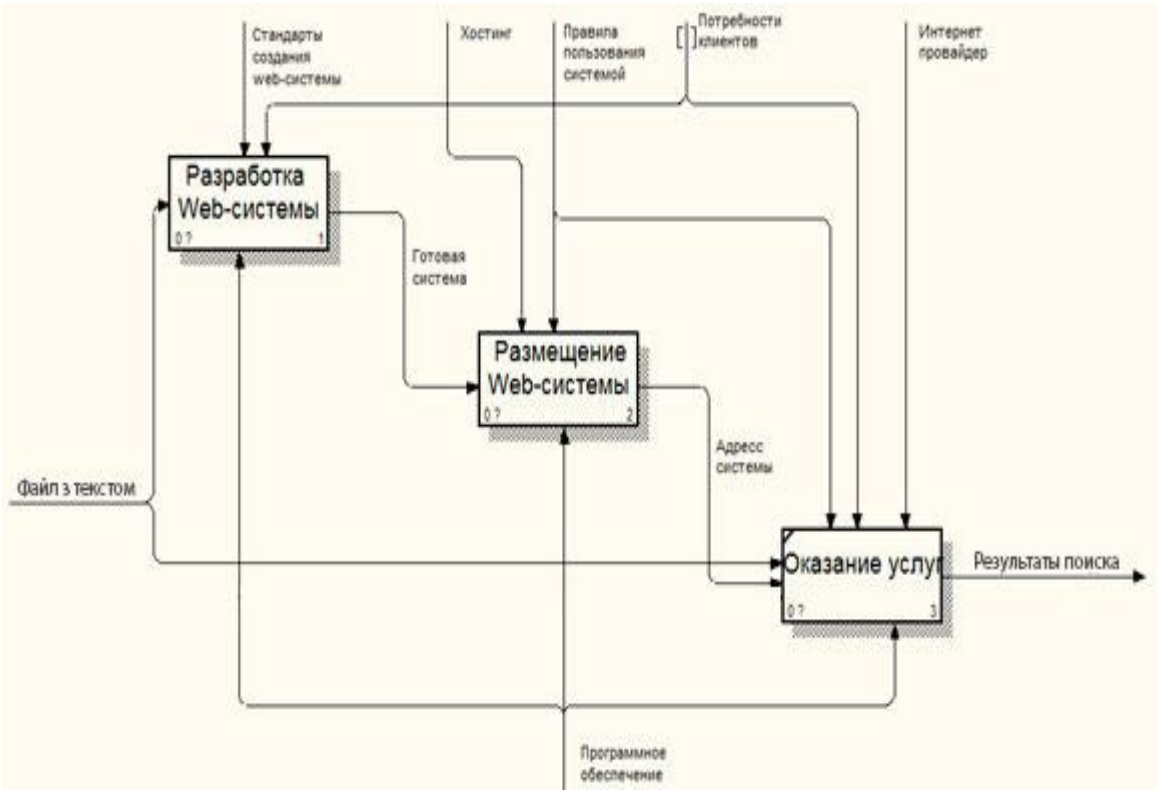

Рисунок 3.2 – Діаграма декомпозиції надання послуг системою

При декомпозиції 1-го блоку, визначено наступні блоки:

- розробка дизайну – цей блок не має входу; управління – потреби клієнтів та стандарти створення web – систем; механізмом є програмне забезпечення; на виході маємо макет системи;
- розробка серверних та клієнтських скриптів – входом є макет системи та данні з файлу; управління – стандарти створення web – систем; механізмом є програмне забезпечення; на виході маємо динамічну систему(без БД);
- розробка БД – вхід – динамічна web-система та данні з файлу; управління – стандарти створення web – систем; механізмом є програмне забезпечення; на виході – готова база даних;
- заповнення БД – входом є готова БД; управління – потреби клієнтів; механізмом є програмне забезпечення; на виході маємо готову систему.

На рис. 3.3 представлена діаграма декомпозиції 1-го блоку.

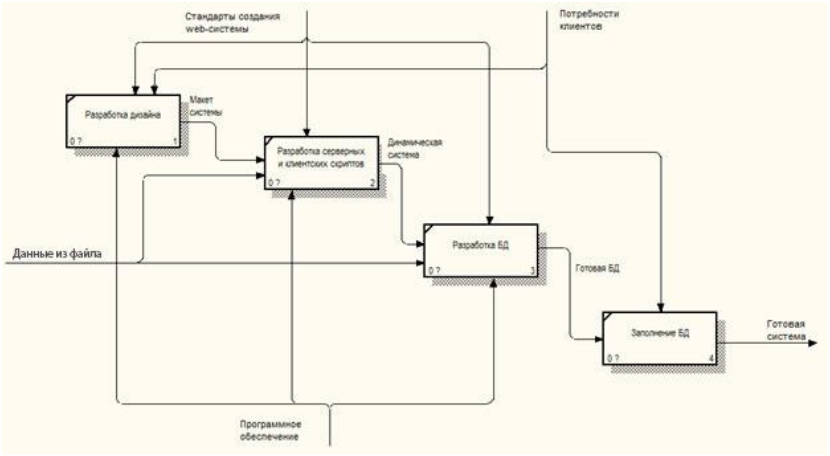

Рисунок 3.3 – Діаграма декомпозиції 1-го блоку

При декомпозиції 2-го блоку, визначено наступні блоки:

– визначення доменного імені – вхід – готова система; управління – хостинг; механізм – програмне забезпечення; вихід – домен;
– визначення зони реєстрації – вхід – домен; управління – хостинг; механізм – програмне забезпечення; вихід – зона реєстрації;
– веєстрація на хостингу – вхід – зона реєстрації; управління – хостинг; механізм – програмне забезпечення; вихід – зареєстрована система;
– перенос системи на хостинг – вхід – зареєстрована система; управління – хостинг та правила користування системою; механізм – програмне забезпечення; вихід – адрес системи.

На рис. 3.4 представлена діаграма декомпозиції 2-го блоку.

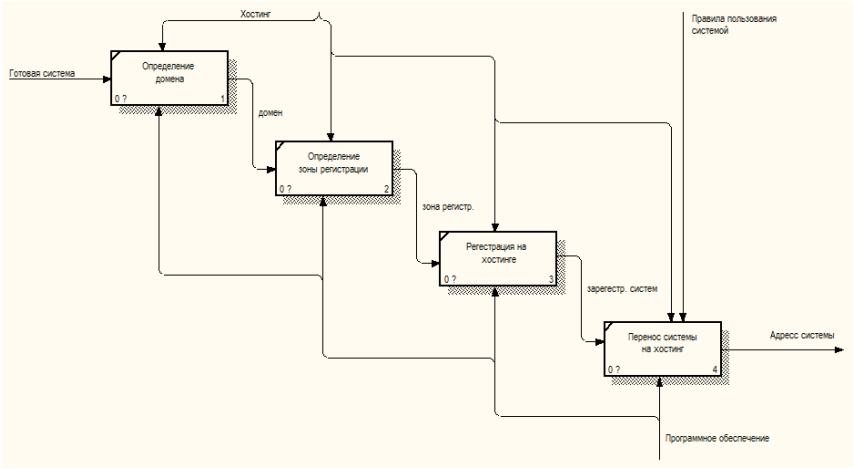

Рисунок 3.4 – Діаграма декомпозиції 2-го блоку

При декомпозиції 3-го блоку, визначено наступні блоки:

- організація роутингу – вхід – адрес системи та клієнти; управління – правила користування системою та Інтернет провайдер; механізм – програмне забезпечення; вихід – зареєстрований клієнт;
- організація роботи з каталогом – вхід зареєстрований клієнт та данні з файлу; управління – Інтернет провайдер та правила користування системою; механізм – програмне забезпечення; вихід – каталог об’єктів;
- організація роботи з системою – вхід – каталог об’єктів та данні з файлу; управління – Інтернет провайдер; механізм – програмне забезпечення; вихід – інструкції;
- формування потреб клієнтів – вхід – карта об’єктів та клієнти; управління – потреби клієнтів Інтернет провайдер; механізм – програмне забезпечення; вихід – результати пошуку.
-

На рис. 3.5 представлена діаграма декомпозиції 3-го блоку.

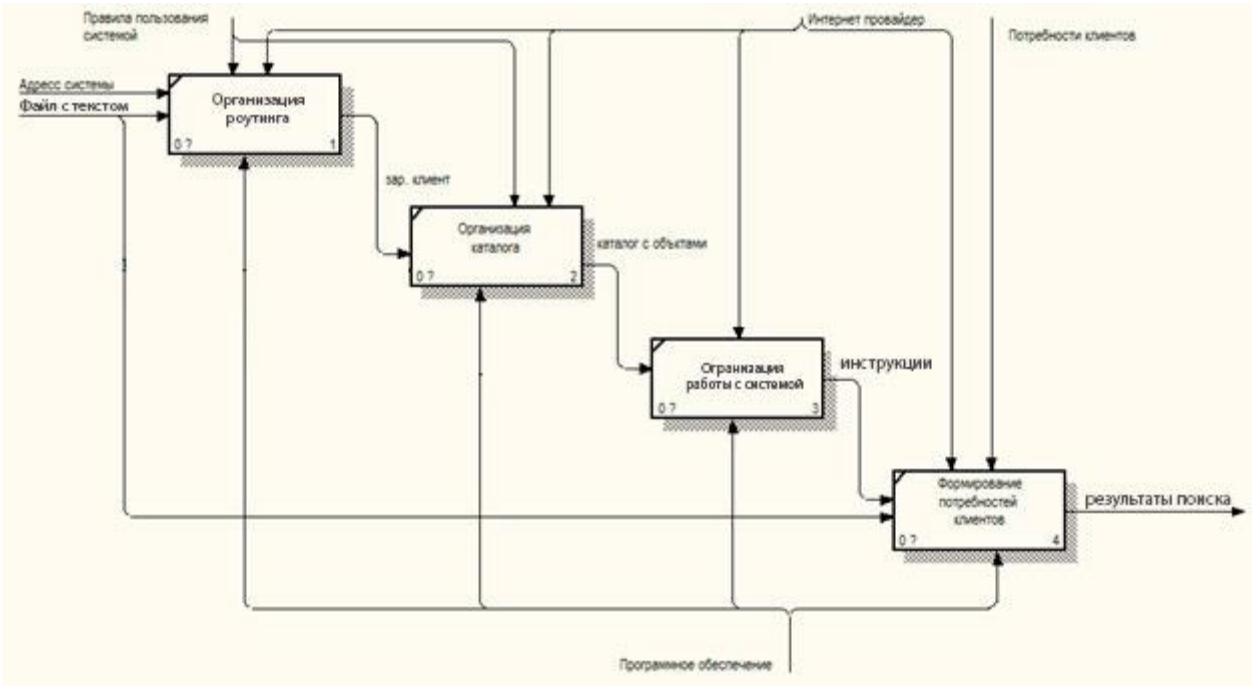

Рисунок 3.5 – Діаграма декомпозиції 3-го блоку

3.2 Проектування інформаційної системи за допомогою стандарту IDEF3(документування технологічних процесів)

IDEF3 є стандартом документування технологічних процесів, що відбуваються на підприємстві, і надає інструментарій для наочного

дослідження і моделювання їх сценаріїв. Сценарієм (Scenario) ми називаємо опис послідовності змін властивостей об'єкта, в рамках розглянутого процесу Виконання кожного сценарію супроводжується відповідним документообігом, який складається з двох основних потоків: документів, що визначають структуру і послідовність процесу (технологічних вказівок, описів стандартів тощо), та документів, що відображають хід його виконання (результатів тестів і експертиз, і т.д.). Для ефективного управління будь-яким процесом, необхідно мати детальне уявлення про його сценарії і структурі супутнього документообігу. Засоби документування та моделювання IDEF3 дозволяють виконувати наступні завдання:

- документувати наявні дані про технологію процесу, виявлення, скажімо, в процесі опитування компетентних співробітників, відповідальних за організацію даного процесу;
- визначати і аналізувати точки впливу потоків супутнього документообігу на сценарій технологічних процесів;
- визначати ситуації, в яких потрібно прийняття рішення, що впливає на життєвий цикл процесу, наприклад зміна конструктивних, технологічних чи експлуатаційних властивостей кінцевого продукту;
- сприяти прийняттю оптимальних рішень при реорганізації технологічних процесів;
- розробляти імітаційні моделі технологічних процесів, за принципом "як буде, якщо ...".

Існують два типи діаграм в стандарті IDEF3, що представляють опис одного і того ж сценарію технологічного процесу в різних ракурсах. Діаграми відносяться до першого типу називаються діаграмами Описи Послідовності Етапів Процесу (Process Flow Description Diagrams, PFDD), а до другого – діаграмами стану об'єктів в і його трансформації Процесі (Object State Transition Network, OSTN). За допомогою діаграм PFDD документується послідовність і опис стадій розробки системи. Саме за допомогою діаграм PFDD буде створена діаграма для Системи виявлення плагіату.

Контекстна діаграма в IDEF3 відображає основну функцію системи. Вона складається з єдиної роботи – «Організація пошуку плагіату через систему».

Контекстна діаграма представлена на рис. 3.6.

Рисунок 3.6 – Контекстна діаграма інформаційно-пошукової системи

Провівши декомпозицію контекстної діаграми, спостерігається послідовність виконання робіт.

На першому місці серед робіт стоїть «Розробка Інтернет системи». Наступною йде робота – «Розміщення Інтернет системи». Ці дві роботи поєднані стрілкою «зв'язок відношень». Це означає, що перша робота не обов’язково повинна завершитись до початку другої. Іншими словами, можна розмістити у мережі незакінчену версію системи, в якій реалізовані тільки основні функції, і потім дороблювати її. Потім, «Розміщення Інтернет системи» поєднується старшим зв’язком з «Реалізація угод». Це означає, що цей блок не почне свою роботу доти, доки попередній не закінчить виконання. Тобто, не можна до повного розміщення системи у мережі забезпечувати її працю з клієнтами і т.д.

Діаграма декомпозиції показана на рис. 3.7.

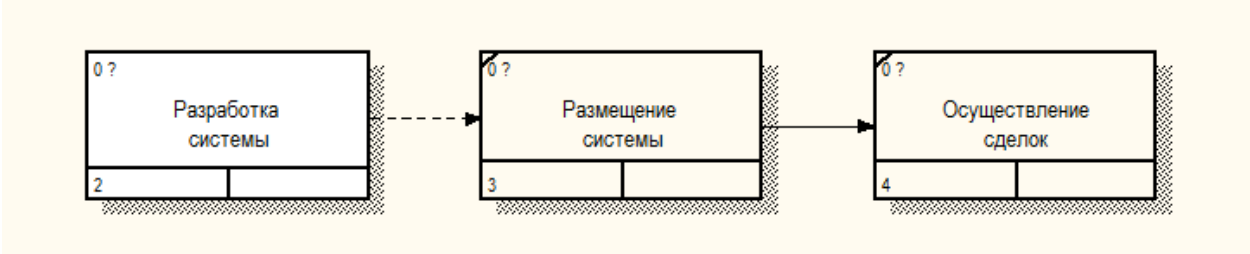

Рисунок 3.7 – Діаграма декомпозиції системи

При декомпозиції 1-го блоку цієї діаграми, можна визначити наступні 5 блоків:

– front-end розробка – розробка, в яку входить написання клієнтських скриптів;
– розробка макету сайту – цей блок включає в себе, створення шаблону у візуальному редакторі, і його подальшу верстку з використанням HTML та CSS;
– back-end розробка – розробка, яка включає в себе написання серверних скриптів;
– створення БД;
– наповнення БД.

Блоки «Front-end розробка», «Розробка макету сайту» та «Back-end розробка» з обох боків об'єднані логічними «Асинхронними И». Це свідчить про те, що ці системи можуть починатися незалежно друг від друга, але для переходу до наступного блоку, всі вони мають буди виконаними. Тільки після цього можна перейти до розробки БД та її наповнення.

Діаграма декомпозиції показана на рис. 3.8.

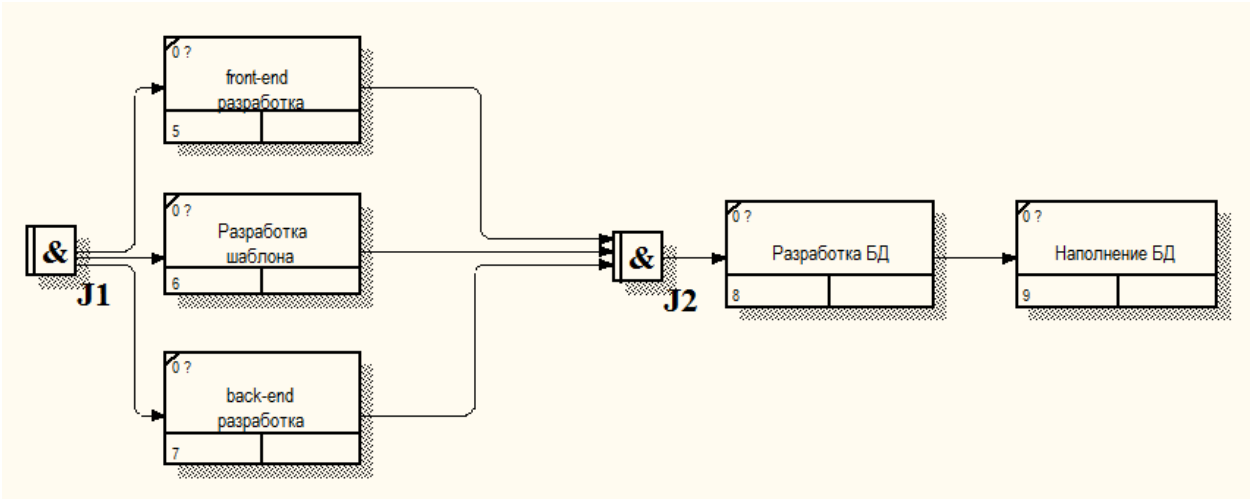

Рисунок 3.8 – Діаграма декомпозиції блоку «Розробка системи»

При декомпозиції блоку «Розміщення системи», можна визначити наступні 4 блоки:
– визначення доменного імені;
– визначення зони реєстрації;
– реєстрація на хостингу;
– перенесення web-системи на сервер.

Усі блоки у цій декомпозиції об’єднані старшим зв’язком. Вони мають виконуватися у строгій послідовності – один за одним.

Діаграма декомпозиції показана на рис. 3.9.

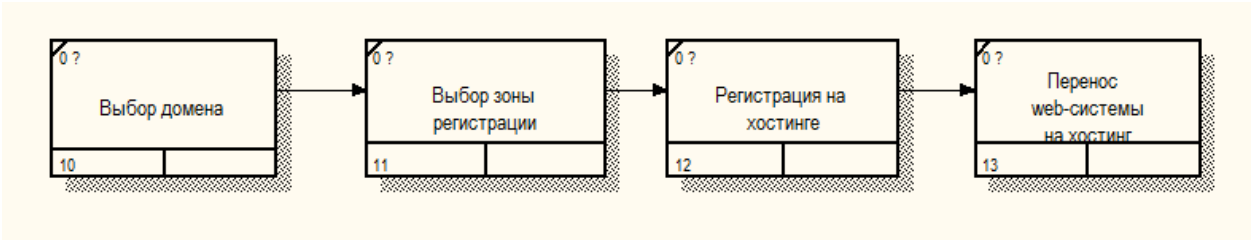

Рисунок 3.9 – Діаграма декомпозиції блоку «Розміщення системи»

При декомпозиції блоку «Здійснення угод», можна визначити наступні 4 блоки:

– реєстрація;
– організація роботи з картою;
– організація роботи з каталогом;
– формування списку потреб клієнтів.

З блоку «Реєстрація» виходить стрілка зв'язок відношень, тобто користувача не обов’язково реєструватися у системі, щоб отримати деякі з послуг. Всі наступні блоки об’єднані старшим зв’язком і можуть бути виконані тільки тоді, коли виконані попередні.

3.3 Проектування навігації по інформаційній системі(карта сайту)

Інформаційне середовище WWW базується на технології гіпертексту, в основі якої лежить концепція зв'язування документів за допомогою посилань. Саме посилання об'єднали Інтернет в єдиний простір, давши користувачам можливість вільно переміщатися, не замислюючись про структуру та просторової розподіленості цієї складної системи.

Правильне використання гіперпосилань є найважливішим чинником підвищення юзабіліті сайту. Відомий фахівець з юзабіліті Якоб Нільсен стверджує, що навігація сайту повинна в будь-який момент надавати відвідувачеві відповіді на три питання:

– Де я знаходжусь?
– Де я вже був?
– Куди я можу піти?

Аналіз шляхів потрапляння відвідувачів на сайт показує, що основним джерелом трафіку є пошукові системи. Тому важливо, щоб відвідувач не тільки перейшов на внутрішню сторінку сайту з пошукової системи, але відразу ж зміг зрозуміти, в якому підрозділі якого розділу і на якій сторінці він знаходиться.

При проектуванні сторінок необхідно враховувати ситуацію, коли користувач потрапляє відразу на внутрішню сторінку сайту. Тому на будь-якій сторінці сайту повинні бути присутніми наступні елементи: логотип компанії, який є посиланням на стартову сторінку, а також короткий опис сфери діяльності центральна навігація, відображає структуру основних інформаційних блоків сайту (розділів і підрозділів) поле пошуку по сайту. Якщо необхідно виключити потрапляння користувача з пошукової системи на певну сторінку, необхідно встановити заборону на її індексування пошуковими системами. Пошукові сервери повинні виключити таку сторінку з баз даних.

Класифікація елементів навігації. Всі засоби навігації діляться по відношенню до сайту на зовнішні і внутрішні. Зовнішні засоби реалізовані у вигляді кнопок браузера, керуючих переходами по сайту.

Бажано, щоб структурні посилання відрізнялися від допоміжних і перехресних посилань, розміщених у тексті. На більшості сайтів вони виглядають однаково. Тому часто буває незрозуміло, веде посилання на цілий розділ або на єдину сторінку. З проблемою визначення вмісту таких посилань часто стикаються користувачі download-менеджерів, коли необхідно вказувати, до якого рівня вкладеності необхідно завантажувати цілий сайт або окремі його розділи.

Нижче представлена схема можливих переходів на різні сторінки у межах системи виявлення плагіату (рис. 3.10).

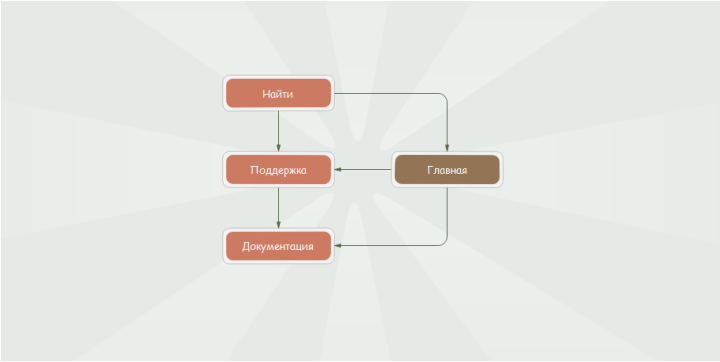

Рисунок 3.10 – Схема можливих посилань web-системи

Кількість переходів не велика. Але цей факт не заважає в повній мірі користуватися усіма функціями системи.

4 ПРАКТИЧНА РЕАЛІЗАЦІЯ

4.1 Огляд структури шаблонів системи виявлення плагіату

Для Web-системи дуже важливий стиль, що додає Інтернет-ресурсу власне обличчя і пізнаваність. Можна виділити наступні елементи, що у створенні стилю:

- шрифт – в межах публікації він повинен мати однакові характеристики (накреслення, висоту, колір);
- колірна схема Wев-сторінок – вибір тих трьох кольорів сторінки, котрі будуть використовуватися для представлення звичайного тексту, посилань і відвіданих посилань. Колірна схема повинна повторюватися на всіх сторінках публікації, це створить у відвідувача відчуття зв'язності системи. Кольори посилань вибирають таким чином, щоб вони були помітні і в той же час не заважали читати основний текст;
- графічне оформлення системи – має укладатися в загальну колірну схему;
- результатом цього етапу проектування повинен бути остаточний ескіз Wев-сторінки.

Професійна розробка відрізняється від аматорської насамперед тим, що всі сторінки Інтернет-системи виконані в єдиному стилі. Щоб витримати стиль, необхідно на початку розробити шаблон сторінки. Шаблони зручні тим, що більшість сторінок верстають за подобою однієї сторінки майже автоматично. Основні елементи, які повинні бути присутніми на сторінках інформаційної системи фармацевтичної фірми:

- назва;
- логотип (графічний знак, який ідентифікує компанію);
- навігаційне меню;
- дані (власне зміст сторінки).

Перш за все, слід приділити увагу сервісному обслуговуванню – зручний каталог продукції, пошук по товарах і послугах, ясні і не складні форми, які необхідно заповнити користувачеві для реєстрації, формуванні замовлення та покупки, контактна інформація. Графічні елементи використовуються не тільки ілюстрації, фотографії продукції, але піктограми, іконки – графічні символи, які допомагають відвідувачеві орієнтуватися в інформаційних обсягах.

Структура шаблону складається з ряду елементів (рис.4.1).

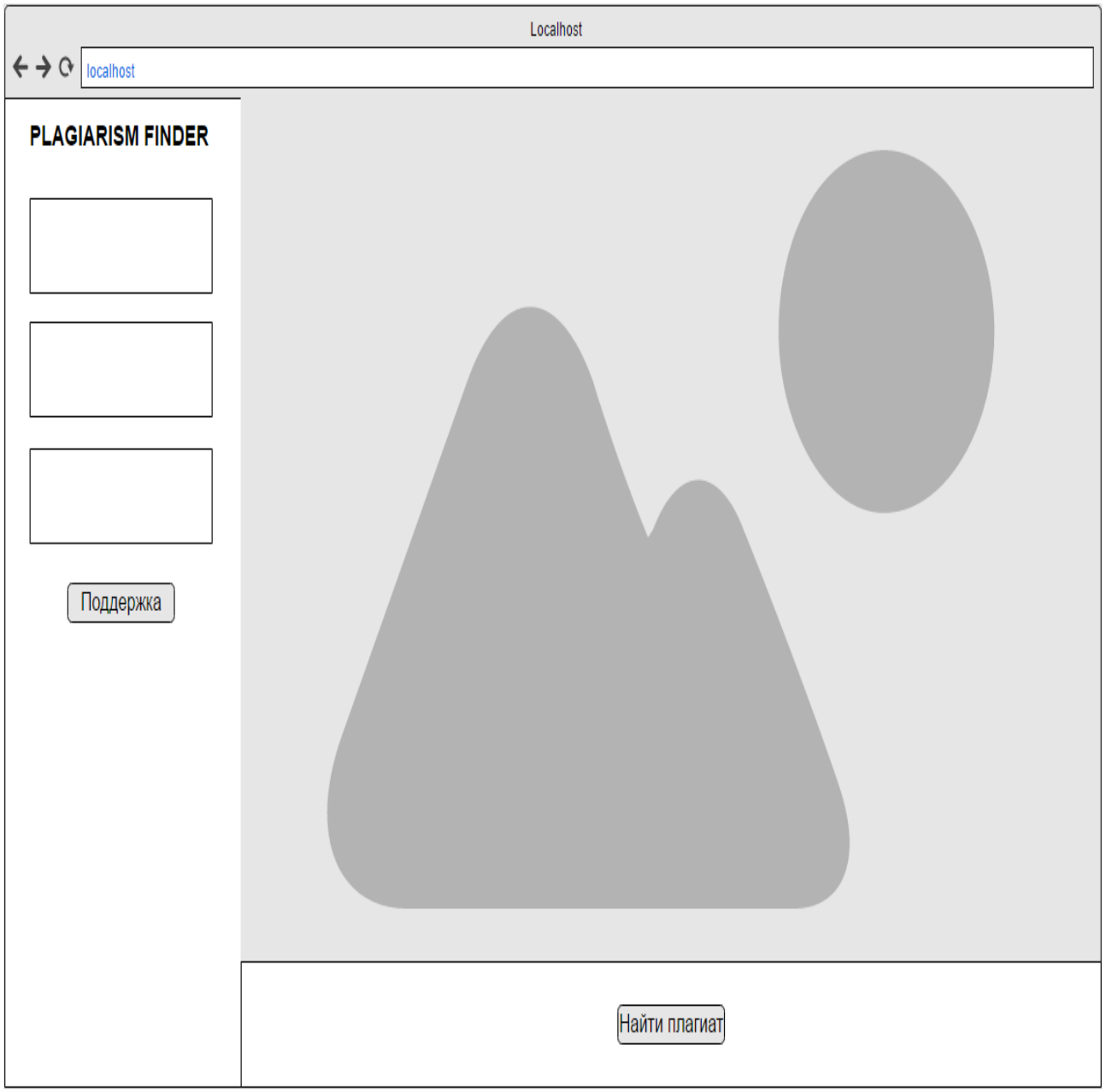

Рисунок 4.1 – Структура головної сторінки Системи антиплагіату

Шаблон дозволяє нам використовувати статичні частини на усіх сторінках ресурсу. Це дуже зручно та економить багато пам’яті. Для створення адаптивного дизайну був використаний TwitterBootstrap Framework(розділ 2.4.5). Завдяки цьому робота з портативних девайсів поліпшується в рази. Це насамперед важило для людей, які працюють в дорозі с мобільного або планшету, та не мають доступу до мережі з великою швидкістю.

Нижче показана структура, яка буде відображатися при використанні мобільного телефону. Схожа структура також буде використовуватися для планшетів (рис. 4.2).

Рисунок 4.2 – Структура головної сторінки системи виявлення плагіату при перегляді с телефону

4.2 Керівництво користувача

Головні сторінки – найбільш дороге майно у світі. Щорічно компанії і приватні особи витрачають достатньо великі кошти на ділянку простору,

який за площею не перевищує декількох квадратних футів. І цьому є пояснення. Головна сторінка робить на прибутку компанії набагато більший вплив, ніж прибутку від електронної торгівлі: головна сторінка – це обличчя компанії, видиме всьому світу. Все частіше і частіше, перед тим, як почати ділові відносини з вашою компанією, ваші потенційні замовники спочатку дивитимуться на ваш сайт, – незалежно від того, чи планують вони укласти з вами угоду через сайт чи ні.

Головна сторінка – найбільш важлива сторінка на більшості сайтів, її переглядають набагато частіше, ніж будь-які інші сторінки сайту. Звичайно, відвідувачі не завжди потрапляють на сайт через головну сторінку. Сайт подібний дому, в якому кожне вікно є одночасно і дверима. Користувачі запросто можуть прийти на сайт з пошукових серверів або з інших сайтів по посиланнях, які ведуть углиб вашого сайту. Проте, одна з перших речей, які відвідувачі роблять після приходу на новий сайт, – це йдуть на головну сторінку. Глибокі посилання дуже корисні, але вони не дають споживачам того загального уявлення про сайт, яке дає головна сторінка. Якщо тільки ця сторінка слідує певним правилам.

Головна сторінка Системи виявлення плагіату зображена на рис. 4.3.

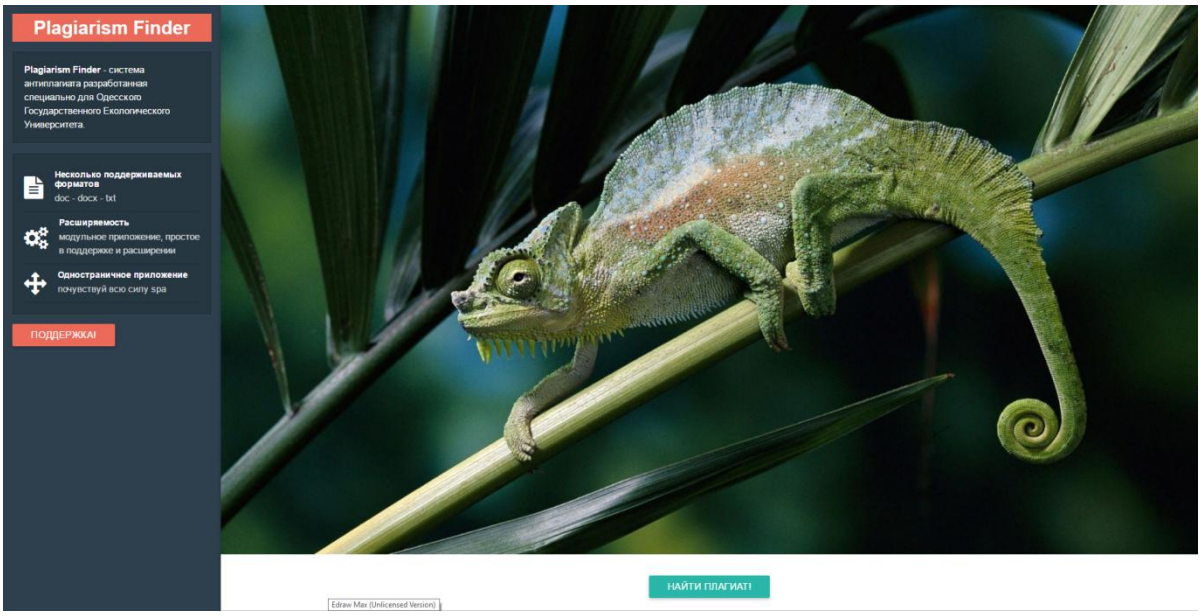

Рисунок 4.3- Головна сторінка web-системи

З головної сторінки, за допомогою посилань, ми можемо далі й далі поглиблюватися систему. Розглянемо основні розділи.

«Найти плагиат» – цей розділ описує компанію. Читаючі інформацію, клієнт може уявити з яким агентством він має справу, варто чи ні користуватися саме цим агентством.

На рис. 4.4 представлений розділ системи – «Найти плагиат»

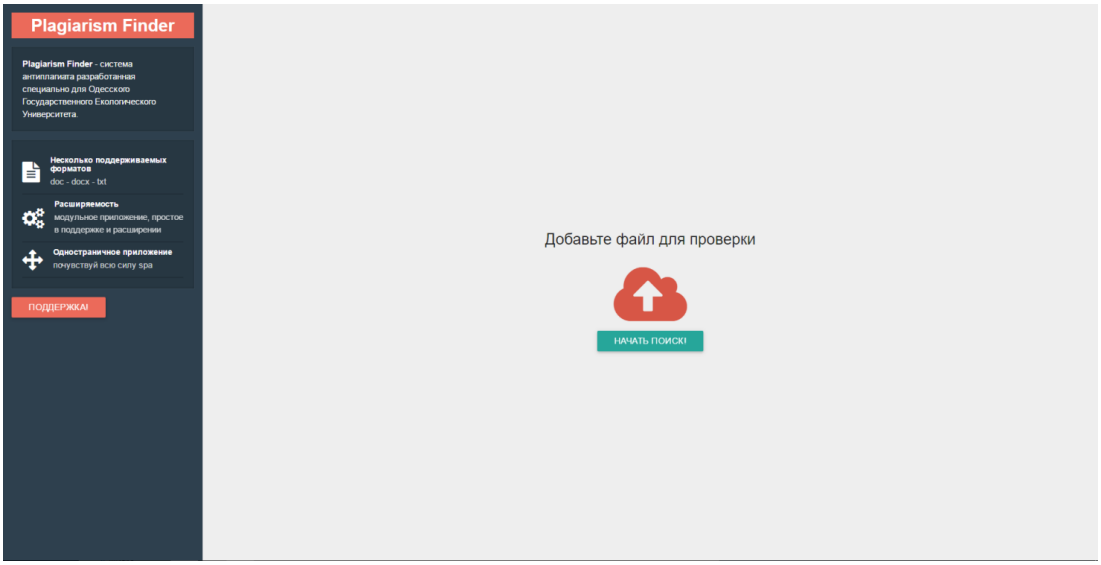

Рисунок 4.4 – Розділ «Найти плагиат» web-системи

«Результат» – цей розділ дозволяє клієнту знайти усю необхідну інформацію, за допомогою якої він може здійснювати зворотній зв'язок з агентством.

На рис. 4.5 представлений розділ системи – «Результат»

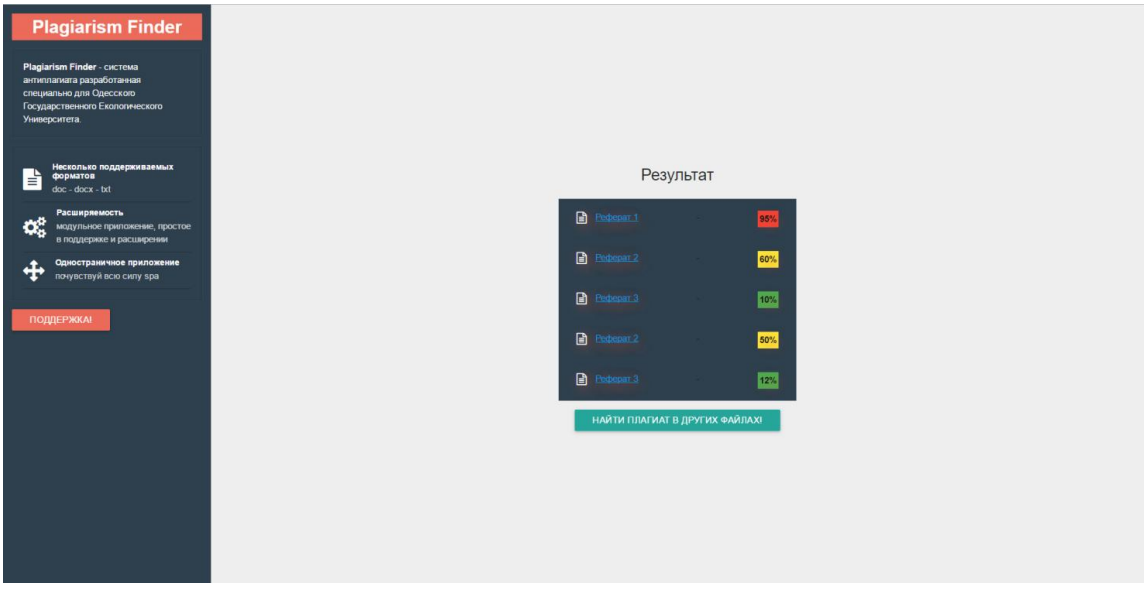

Рисунок 4.5 – Розділ «Результат» web-системи

ВИСНОВКИ

В ході виконання магістерської роботи було реалізовано систему виявлення плагіату, яка побудована за принципами SPA. Основна мета досягнута, так як в системі передбачено завантаження файлів у текстових форматах до серверу, та отримання результатів по відсотку схожості із файлами які розташовані у сховищі.

В роботі були використані найсучасніші підходи, щодо розробки інформаційних систем. Це дозволило нам отримати зручний у використанні продукт, який максимально наближений до настільних додатків, що значно підвищило продуктивність роботи з системою.

У систему виявлення плагіату було закладено модульний підхід, це дозволить нам у майбутньому досить легко модернізувати та підтримувати даний додаток.

ПЕРЕЛІК ПОСИЛАНЬ

1. Сайт компанії PlagScan [Електроний ресурс] – Режим доступу: http://www.plagscan.com/
2. Сайт компанії Unplag [Електроний ресурс] – Режим доступу: http://www.unplag.com/
3. Сайт компанії Антиплагиат [Електроний ресурс] – Режим доступу: https://www.antiplagiat.ru/
4. Офіційна документація React.js [Електроний ресурс] – Режим доступу: https://facebook.github.io/react/docs/
5. Офіційна документація Redux [Електроний ресурс] – Режим доступу: https://github.com/reactjs/redux
6. Офіційна документація методології БЕМ [Електроний ресурс] – Режим доступу: https://ru.bem.info/
7. Офіційна документація NodeJS [Електроний ресурс] – Режим доступу: https://nodejs.org/
8. Ситник В.Ф. та ін. Основи інформаційних систем: Навч. Посібник – 2-е вид., перероб. і доп. – К.: КНЕУ, 2001. – 420 с.
9. Кристофер Косентино. CSS: Web-профессионалам. К.: BHV, 2001. – 300 с.
10. Хокинс С. Администрирование Web-сервера NodeJs и руководство по электронной коммерции – М.: Вильямс, 2001. – 336 с.
11. Бесплатная коллекция уроков для создания сайтов [Електронный ресурс] – Режим доступа: http://ajaxs.ru/
12. Л. Аргерих, В. Чой, Д. Коггсхол и др. JavaScript. Профессиональное рограммирование – СПб.: Символ, 2003. – 1048 с.
13. И. Браун. Веб-разработка с применением Node и Express – СПб.: Питер, 2016. – 336 с.
14. Даккет Д. HTML и CSS. Разработка и дизайн веб-сайтов. Пер. з англ. – М.: Эксмо, 2013. – 480 с.
15. Хокинс С. Администрирование Web-сервера NodeJs и руководство по электронной коммерции – М.: Вильямс, 2001. – 336 с.