МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційне та програмне забезпечення форуму для автомобілістів. Серверна частина.»

Завідувач випускаючої кафедри — **Довбиш А. С.**

Керівник роботи — **Шелехов І. В.**

Студент групи Ін-73 — **Стовпак Н.І.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи Ін-73 спеціальності "Комп'ютерні науки" денної форми навчання Стовпака Нікіти Івановича.

Тема: «Інформаційне та програмне забезпечення форуму для автомобілістів. Серверна частина.»

Затверджена наказом по СумДУ
№_____ від _____2021 р.

Зміст пояснювальної записки: 1) аналітичний огляд предметної області веб-форумів; 2) постановка завдання й формування завдань реалізації; 3) характеристика принципів архітектури серверного Веб-додатку; 5) розробка бази даних і серверної частини веб-додатку; 6) аналіз результатів.

Дата видачі завдання "_____"_____20__ р.
Керівник випускної роботи _____ Шелехов І.В.
Завдання прийняв до виконання _____ Стовпак Н.І.

РЕФЕРАТ

Записка: 65 стор., 31 рис., 1 табл., 2 додатки, 18 джерел.

Об’єкт дослідження – серверна частина Web-форуму для автомобілістів.

Мета роботи – метою дипломного проекту є створення серверної частини Web-форуму для обміну інформації та корисних порад між людьми.

Методи дослідження – методами дослідження було обрано JavaScript, Node.js, Express, MySQL.

Результати – було спроектовано та розроблено серверну частину Web-форуму, для комфортного спілкування осіб зі схожими інтересами. Після інтеграції серверної частини користувач має змогу :

- створювати та видаляти власні теми;
- коментувати , видаляти та редагувати власні коментарі;
- обирати найпопулярніші теми
- редагувати персональні дані
- змінювати пошту
- змінювати пароль
- відновлювати пароль

Дану частину було розроблено з застосуванням лише серверних технологій.

WEB-ФОРУМ, JAVASCRIPT, FRONTEND, NODE, MYSQL, EXPRESS

ЗМІСТ

ВСТУП

У сучасному світі веб-сайт – це найефективніша форма електронного маркетингу, яка дає можливість користувачу або компанії відобразити себе так, як вони потребують відповідно до їх цілей. Веб-сайт дає спроможність створення платформи для реалізації своїх ідей, ефективного збуту послуг чи продуктів.

Взаємодія користувачів та компаній в інтернеті є обов'язковою вимогою для будь якого бізнесу в сфері інформаційних технологій. Завдяки соціальним мережам, незалежно від бренду та масштабу, компанії можуть підтримувати зв'язок та спілкуватися з користувачами та клієнтами. Проте сама суть платформи соціальної мережі має недоліки з точки зору комунікацій між компаніями то користувачами. Хоч соціальні мережі і являються важливим каналом спілкування з клієнтами, але формат форуму, де є можливість створити більш спеціалізовану та обширну платформу для більш детального розуміння клієнтського мислення.

Взаємодія з клієнтами відіграє таку ж важливу роль, як і обслуговування клієнтів. Це формує довірчі і довгострокові відносини. Забезпечуючи більш ніж двостороннє спілкування, форум може створити активну онлайн-спільноту, яка не тільки взаємодіє з контентом компанії, але і дозволяє користувачам взаємодіяти один з одним.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАДАЧІ

1.1 Особливості предметної області веб-форуму

Веб-форум або просто форум — інтернет-ресурс, популярний різновид спілкування в інтернеті. На форумі створюються теми для спілкування, що робить його кращим за чат. Всі, кого цікавить певна інформація, можуть зручно й швидко переглянути її на форумі. Форуми можуть бути присвячені програмному забезпеченню, автомобілям, футбольній команді тощо [1].

Більшість форумів дозволяють анонімним користувачам переглядати повідомлення на форумі, але вимагають, щоб ви створили обліковий запис, для публікацій повідомлень на форумі. Публікуючи повідомлення на форумі, користувач може створювати нові теми або публікувати відповіді в рамках існуючих тем. Оскільки веб-форуми складаються з користувацького контенту, вони(форуми) продовжують рости, поки користувачі відвідують сайт і публікують повідомлення.

Веб-майстру веб-форуму просто потрібно управляти форумом, що може зажадати переміщення, об'єднання та архівування тем. Це також може включати в себе моніторинг публікацій і видалення недоречних. Оскільки веб-форуми постійно ростуть, вони стали значною частиною Інтернету. Фактично, якщо користувач бажає переглянути інформацію з певної теми, є велика ймовірність, що одна або кілька сторінок форуму з'являться в перших результатах пошукової системи. Зрештою, якщо у користувача є питання про щось, швидше за все, це не унікальний випадок. Користувач може використовувати форуми, щоб почерпнути інформацію від тих, хто вже ставив схожі або ідентичні питання в минулому. І навпаки, користувач може допомогти іншим, поділившись своїми ідеями і відповідями на онлайн-форумі.

- Форум ділиться на безліч категорій, і кожна категорія може мати безліч під-форумів.
- Під-форуми також можуть мати додатковий під-форум.
- Теми - це місце, де учасники можуть публікувати свій контент.

- Відображення ланцюжків можна налаштувати так, щоб вони відображалися у вигляді вкладених повідомлень з відповіддю на відповідь або просто повідомлень, починаючи з найостанніших і закінчуючи самими старими.
- Пост(повідомлення) - це останній елемент, але по суті це зміст форуму [2].

Користувачі можуть висловлювати свої думки в редакторі і розміщувати їх в гілці. Зазвичай повідомлення показує всі відповідні відомості про користувача, який його розмістив, так що учасники форуму можуть напряму зв'язатися з користувачем.

Нижче наведено приклад структури типового сайту онлайн-форуму, що містить різні елементи.

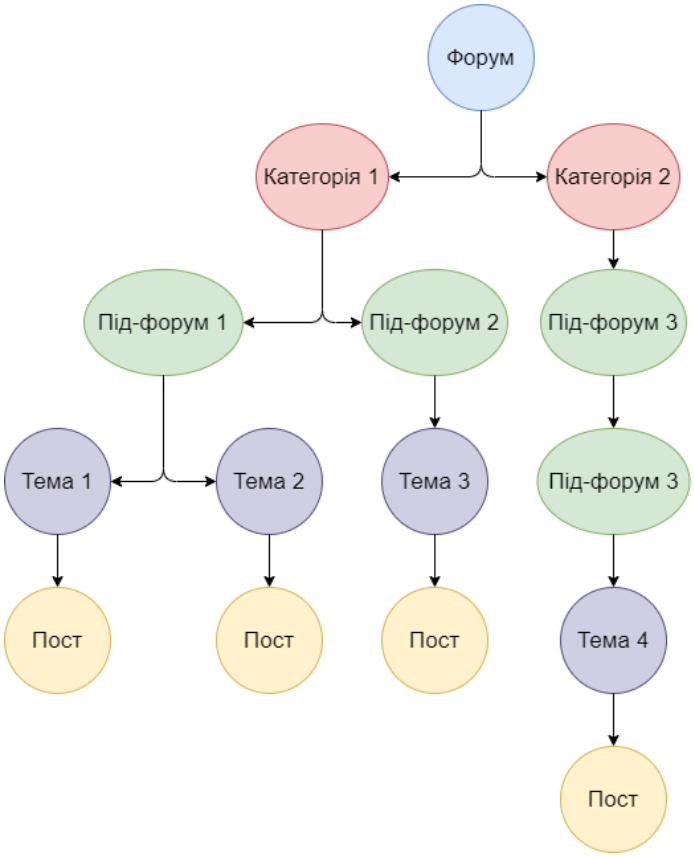

Рисунок 1.1 – Структура сайту

1.2 Аналіз аналогів

Перед тим, як створювати власний веб-форум, потрібно провести аналіз аналогів, які вже існують на ринку для отримання детальної інформації про них.

Цей етап є невід’ємною частиною проектування веб-додатку для створення в результаті конкурентно спроможного веб-форуму.

При подальшому аналізі аналогів будемо розглядати тільки технічну частину веб-додатків. Дизайн, спосіб взаємодії користувача та інше, що зв’язано з front-end розробкою в даній роботі розглянуті не будуть.

Виберемо 3 веб-форуми, які є найпопулярнішими на тематику автомобілів. В результаті маємо такий список:

- https://forums.drom.ru/
- http://forum.autoua.net/
- https://autoforum.info/

Виділимо основні категорії в рамках яких буде проведений аналіз та сформуємо список.

- Пошукові системи
    - Індексація
    - Перевірка на віруси
- Трафік
    - Перегляди
    - Користувачі
    - Рейтинг Alexa
- Посилання на сайт
    - Посилаються сторінки
    - Посилаються домени
    - Довіра сайту(Trust Rank & Domain Rank)
- Соціальні мережі
    - Facebook
    - Вконтакте
    - Twitter
- Оптимізація
    - Заголовок сторінки

- Опис сторінки
- Розмір завантажених ресурсів
- Швидкість завантаження html
- Внутрішні посилання
- Помилки html коду

- Юзабіліті
  - Код повернення неіснуючої сторінки 404
  - Посилання на сторінку 404
  - Швидкість завантаження від Google
    - Десктоп
    - Мобільні пристрої
  - Кеш браузера

Для загального аналізу аналогів створимо таблицю. Проаналізувавши дані зробимо висновки щодо кожного з аналогів. Отримані дані використаємо для проектування та реалізації свого веб-форуму.

Таблиця 1.1 – Аналіз аналогів

Характеристика	forums.drom.ru	forum.autoua.net	autoforum.info
Індексація Google	961 000	184 000	1 790
Перевірка на віруси	Не виявлено	Не виявлено	Не виявлено
Перегляди за місяць	14 080 000	2 060 000	12 700
Користувачі за місяць	3 740 000	386 000	6 700
Рейтинг Alexa	925	209 088	2 280 760
Посилаються сторінки	954 275	561 857	5 362
Посилаються домени	8 761	1 307	125
Довіра сайту(Trust Rank & Domain Rank)	Trust Rank – 35 Domain Rank - 42	Trust Rank – 21 Domain Rank - 29	Trust Rank – 0 Domain Rank - 11
Facebook	Посилання відсутнє	Посилання існує	Посилання відсутнє
Вконтакте	Посилання відсутнє	Посилання відсутнє	Посилання відсутнє

Продовження табл. 1.1.

Характеристика	forums.drom.ru	forum.autoua.net	autoforum.info
Twitter	Посилання відсутнє	Посилання існує	Посилання відсутнє
Заголовок сторінки	Форумы об автомобилях в России	Автомобильный форум. Форум помощи выбора, покупки и продажи авто | Autoua.net	Autoforum.info – живое общение всех автолюбителей
Опис сторінки	Автомобильные форумы на Drom.ru	Форум Автоуа - все про автомобильную жизнь. Первый украинский автомобильный форум. Кладезь ценнейшей информации и полезных советов для всех сфер автомобильной и не только автомобильной жизни.	Форум автомобилистов России. Ремонт, техническая поддержка, тюнинг автомобилей. Общаемся и делимся опытом!
Розмір завантажених ресурсів	648 КБ	1582 КБ	1405 КБ
Швидкість завантаження html	1,29 секунди	0,54 секунди	1,03 секунди
Внутрішні посилання	580 (індексуються 481)	194	384
Помилки html коду	154	937	4
Код повернення неіснуючої сторінки 404	Отриманий код 200	Отриманий код 404	Отриманий код 404
Посилання на сторінку 404	Посилання існує	Посилання існує	Посилання існує
Швидкість завантаження від Google(Десктоп)	3,6 секунди	4,1 секунди	2,3 секунди
Швидкість завантаження від Google(Мобільні пристрої)	7,2 секунди	15,2 секунди	5,1 секунди
Кеш браузера	42 ресурси без заголовків кешування	Коректно налаштоване	56 ресурси без заголовків кешування

Кожен з аналогів має специфічні помилки та властивості. Для більш повного розуміння аналізу веб-форумів опишемо кожен з них індивідуально.

forums.drom.ru

Найпопулярніший форум з представлених має тільки не критичні помилки, але навіть вони впливають на роботу сервісу для користувача. Було знайдено такі помилки:

- Відсутність перенаправлення з домену www.
- Код відповіді на не існуючу сторінку повинен повертати код 404, але повертає код 200.
- Тривалий час завантаження крутного контенту, через що сайт завантажується відносно довго як з десктопу, так і з мобільного простою.

forum.autoua.net

Даний форум також має велику кількість переглядів та користувачів, проте було виявлено декілька критичних та не критичних помилок, а саме:

- Домен сайту знаходиться в реєстрі заборонених сайтів.
- Відсутня система статистики.
- 338 зовнішніх посилань, коли рекомендована їх кількість до 100.
- Сайт не доступний по протоколу https. Відсутній SSL протокол.
- Низька швидкість завантаження сторінки з десктопу та критично низька швидкість для завантаження з мобільних пристроїв.
- Відсутня адаптивність для мобільних пристроїв.

autoforum.info

Даний форум не є популярним. Із-за маленької кількості користувачів, серверна частина додатку працює коректно. На даній платформі відсутні критичні помилки, є тільки певні попередження, які майже не впливають на якість роботи форуму. Головною проблемою даного сайту є SEO частина, проблема с кількістю контенту та користувачів.

1.3 Постановка задачі

Веб-форуми є актуальною платформою для користувачів на сьогодні. За допомогою форуму користувач може знайти бажану інформацію на будь-яку тему. Тема автомобілів з часом тільки поширюється, так як все більше людей стають власниками авто. Люди, які тільки придбали авто не володіють досвідом та знаннями про автомобілі в повній мірі. Веб-форум на авто тематику також буде корисним і для досвідчених водіїв, які можуть або ділитися своїм досвідом, або заповнювати деякі пробіли в своїх знаннях. Проаналізувавши предметну область та аналоги, які є на ринку, можна стверджувати про релевантність створення веб-форуму на авто тематику та конкурентоспроможну платформу, яка не буде мати недоліків, які були виявлені при аналізі існуючих аналогів.

Для реалізації серверної частини веб-форуму, яка є невід’ємним структурним модулем веб-додатку необхідно виконати такі вимоги:

1. Проектування бази даних.
2. Проектування та реалізація серверної архітектури.
3. Коректна взаємодія серверної частини з клієнтською.
4. Тестування веб-форуму.

2 АРХІТЕКТУРА СЕРВЕРНОЇ ЧАСТИНИ ВЕБ-ДОДАТКУ

Фронтенд – це частина веб-додатку, роль якого полягає в відображенні контенту. Ця частина може мати різні імплементації, такі як: веб-додаток, додаток для десктопу або для мобільних пристроїв. У не залежності від реалізації фронтенду потрібні дані, які він буде опрацьовувати. Без цих даних його можна вважати марним. Між цими частинами існує чітка межа. Бекенд – це частина додатку за допомогою якого додаток може працювати з базами даних, застосовувати до цих даних певні бізнес-правила і відправляти результати роботи на сторону фронтенду, який в свою чергу прийме дані та згенерує інтерфейс, за допомогою якого користувач і буде взаємодіяти з системою [3].

2.1 Аналіз архітектурних реалізацій

Можна стверджувати, що концептуально фронтенд залежить від даних, які він спроможний отримати тільки с бекенду. Наприклад веб-сайт, який побудований за клієнт-серверною архітектурою. Користувач сайту спочатку звертається до серверу, де бекенд опрацьовує запити користувачів. Випадок, коли фронтенд повинен бути опрацьований на стороні сервера називається SSR(Server Side Rendering) [4].

Ідеальний світ для бекенд виглядає приблизно так: на вхід додатків надходять HTTP-запити з даними, на виході отримаємо відповідь з новими даними в зручному форматі. Наприклад, JSON. HTTP API легко тестувати, зрозуміло, як розробляти. Однак є сценарії, коли одного API недостатньо.

Сервер повинен відповідати готовим HTML, щоб повернути краулеру пошукової системи, віддати дані з метатегами для вставки в соціальну мережу або, що ще важливіше, прискорити відповідь на слабких пристроях.

У сучасному Інтернеті є JSX, який зарекомендував себе з відмінної сторони. Існують як плюси, так і мінуси про які довгий час ведуться диспути, проте тільки одне заперечувати не можна - в серверному рендері не обійтися без JavaScript-коду [5].

Реалізація SSR силами бекенд-розробки потрібна в таких випадках:

- Змішуються зони відповідальності. Бекенд програмісти починають відповідати за відображення.
- Змішуються мови. Бекенд програмісти починають працювати з JavaScript.

Найкращим варіантом є відокремити SSR від бекенд. У найпростішому випадку потрібно використовувати JavaScript runtime, встановлювати на нього персоналізоване рішення або фреймворк (Next, Nuxt і т.д.), що працює з JavaScript шаблонізатором, і пропускати через нього трафік, що є загальноприйнятим рішенням в сучасному світі.

2.1.1 Робота з даними

Популярним рішенням для взаємодії з даними є створення універсальних API(Application Programming Interface). Цю роль найчастіше бере на себе API Gateway, що вміє працювати з безліччю мікросервісів. В даному підході є проблеми.

По-перше, проблема команд і зон відповідальності. Сучасний додаток можуть розробляти декілька команд. Кожна команда сконцентрована на своєму бізнес-домені, який має свій мікросервіс для бекенд і свої рендери на стороні клієнта. Припустимо, що клієнтські відображення повністю розділені і є міні-SPA (Single Page Application) в рамках одного великого сайту [6].

У кожній команді є фронтенд і бекенд-розробники. Кожен працює над своїм додатком. API Gateway може стати проблемою із-за відсутності відповідальності за цей модуль між розробниками.

По-друге, проблема надлишкових і недостатніх даних. Розглянемо випадок, коли два різних фронтенда використовують один універсальний API.

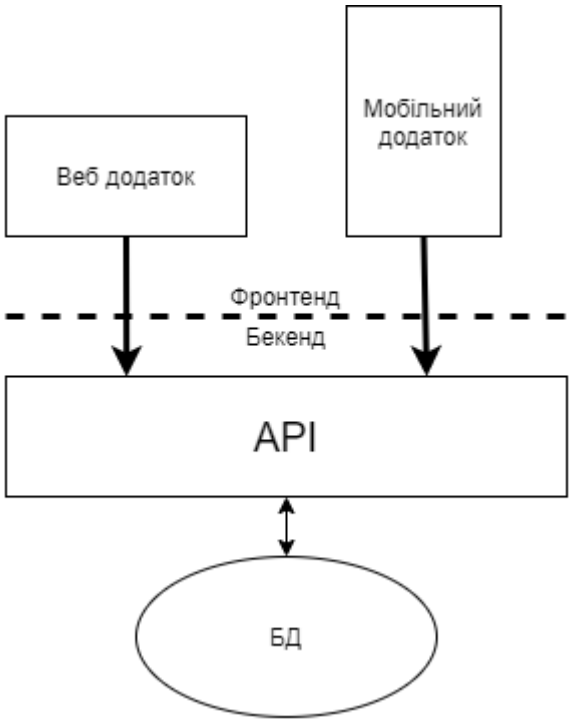

Рисунок 2.1 – Універсальний API

Ці два фронтенда сильно відрізняються. Їм потрібні різні набори даних, у них різний цикл розробки. Варіативність версій мобільного фронтенду максимальна, тому розробники змушені проектувати API з максимальною зворотною сумісністю. Варіативність веб-клієнта низька, фактично розробники повинні підтримувати тільки одну попередню версію, щоб знизити кількість помилок в момент релізу. Якщо «універсальний» API обслуговує тільки веб-клієнтів, все одно потенційно виникає проблема надлишкових або недостатніх даних.

Кожному рендеру потрібен окремий набір даних, отримати який бажано одним оптимальним запитом. У такому випадку універсальний API не може бути використаний, бо доведеться розділити інтерфейси. Потрібно створити свій API Gateway для кожного інтерфейсу.

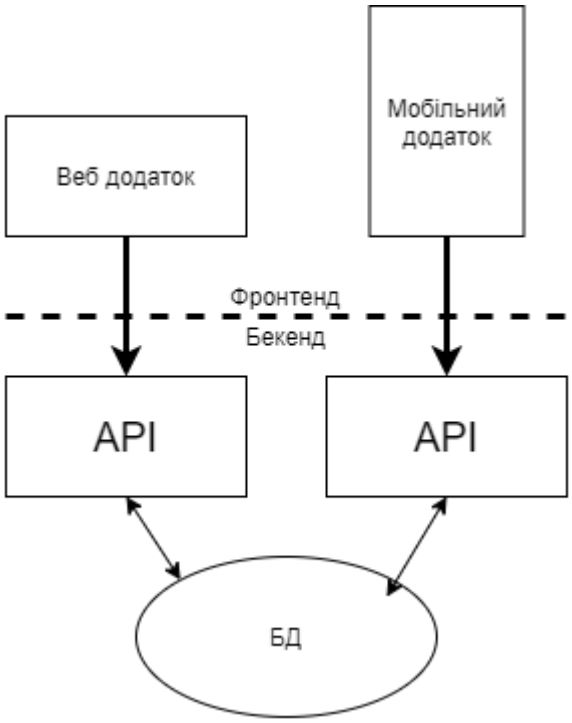

Рисунок 2.2 – Окремі API

Створення такого API ефективніше реалізує команда фронтенда розробки. Всі модулі будуть інтегровані в один серверний додаток. До того ж, контролюючи SSR, бекенд розробники зможуть покласти всі необхідні первинні дані на сторінку в момент рендеру, не роблячи додаткових запитів на сервер. Така архітектура називається Backend For Frontend або BFF [7]. Ідея полягає в наступному: на сервері з'являється новий додаток, який опрацьовує запити клієнта, «опитує» бекенд і повертає оптимальну відповідь.

Універсальний API.

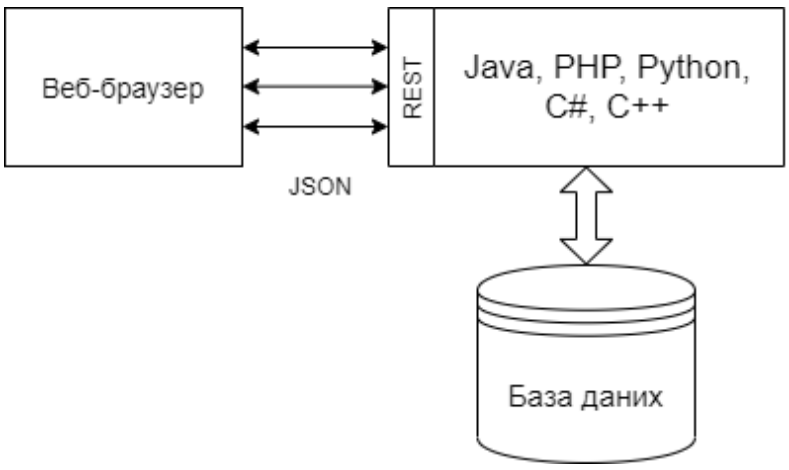

Рисунок 2.3 – Архітектура при універсальному API

API для кожної реалізації фронтенду.

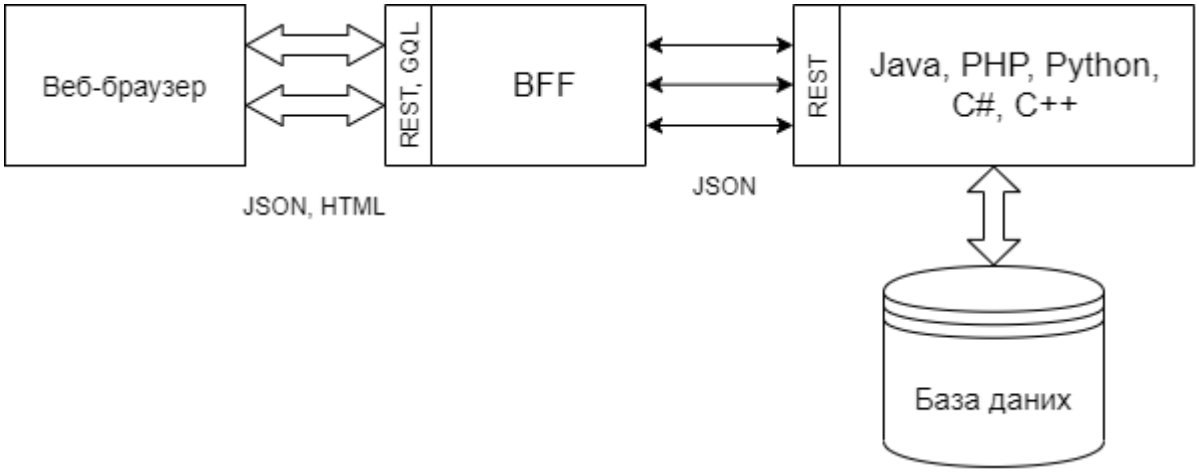

Рисунок 2.4 - Архітектура при окремому API

Для реалізації BFF використовується «Node.js». Для написання коду на клієнтській и серверних частинах потрібно використовувати одну мову програмування. Це дасть можливість перевикористовувати досвід фронтенд розробників для роботи з шаблонами. Архітектура Node.js дає можливість використовувати однопотоковий інтерпретатор JavaScript в комплексі з бібліотекою асинхронного вводу/виводу libuv, яка в своїй реалізації використовує потокові пули. Додаткові обчислення на стороні JavaScript забезпечують ефективність систем.

У базовому випадку Node.js не підходить для операцій, навантажувальних центральних процесорів, однак відмінно працює з асинхронним вводом / виводом, забезпечуючи високу продуктивність. В результаті отримана система може завжди швидко відповісти користувачеві, незалежно від того, наскільки навантажений обчисленнями бекенд. Опрацювати цю ситуацію можливо, миттєво повідомляючи користувача про необхідність почекати завершення операції.

2.1.2 Бізнес-логіка

В системі є 3 великі модулі: бекенд, фронтенд і BFF, який є проміжним елементом між ними. На цьому етапі архітектор повинен вирішити де буде розташовано бізнес-логіка.

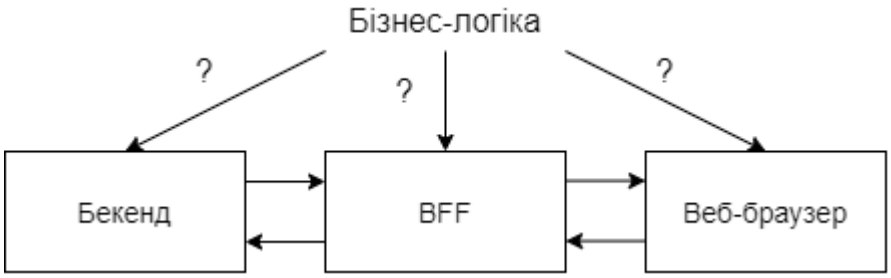

Рисунок 2.5 – Бізнес-логіка

Бізнес-правила не повинні бути розміщені на кожному модулі системи. Для високорівневих політик бізнес логіки потрібно вибрати найбільш близької до даних систему, а саме бекенд.

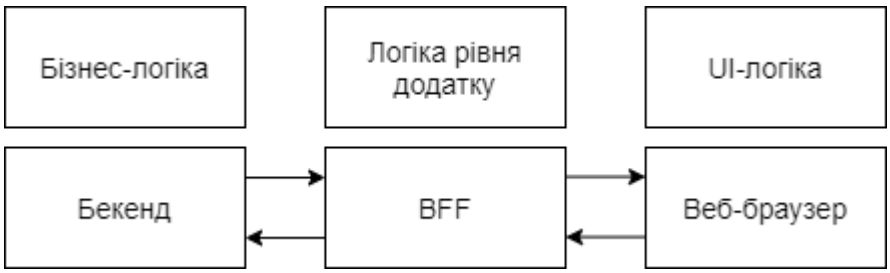

Рисунок 2.6 – Бізнес-політика для бекенду

В реаліях розробки дана схема не завжди буде оптимальним рішенням для системи. Бувають ситуації, коли певну бізнес-задачу оптимально реалізувати на рівні BFF. В таких випадках розробник повинен іти на компроміс.

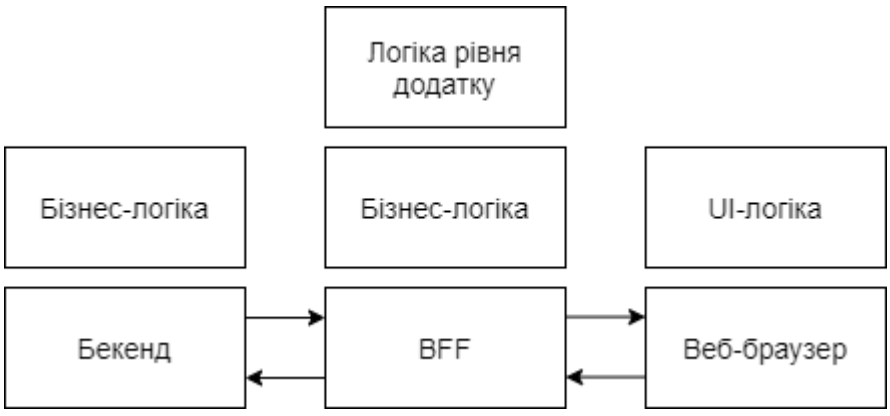

Рисунок 2.7 - Бізнес-політика для BFF

Для створення ідеальної структури системи можна відмовитись від BFF і реалізувати комплексну систему бекенду за допомогою Node.js.

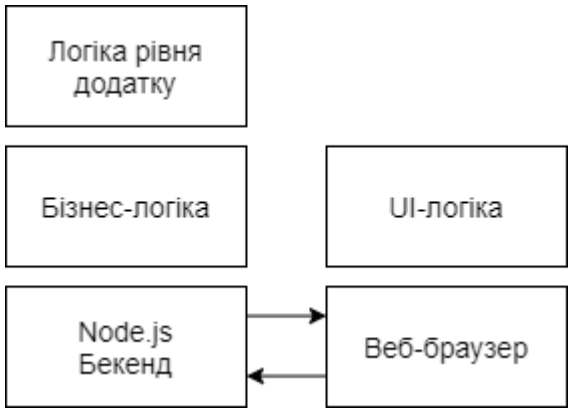

Рисунок 2.8 – Node.js для реалізації бізнес-правил

На даній схемі також з’являються певні нюанси. Можуть існувати бізнес-правила, перенесення яких на сервер приведе до погіршення роботи інтерфейсу. Логіка рівня додатку «проникне» на клієнтську частину: в сучасних SPA вона знаходиться між клієнтом і сервером навіть в разі, коли є BFF.

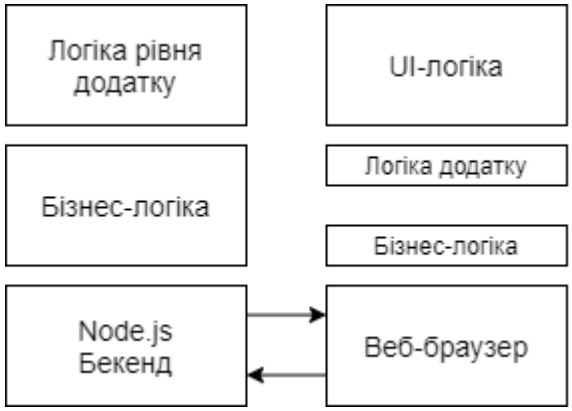

Рисунок 2.9 Проблема проникнення бізнес-логіки

Можна підсумувати, що в будь-якому випадку бізнес-логіка буде реалізовуватись в API Gateway на Node.js.

2.1.3 Імплементація

Найпопулярніше рішення для Node.js-додатків в останні роки - Express. Це перевірений фреймворк, але занадто низькорівневий і не пропонує хороших архітектурних підходів. Основний патерн - middleware. Типовий додаток на Express:

```
const express = require('express');
const app = express();
const {createReadStream} = require('fs');
const path = require('path');
const Joi = require('joi');
app.use(express.json());
const schema = {id: Joi.number().required() };

app.get('/example/:id', (req, res) => {
    const result = Joi.validate(req.params, schema);
    if (result.error) {
        res.status(400).send(result.error.toString()).end();
        return;
    }
    const stream = createReadStream( path.join('..', path.sep,
`example${req.params.id}.js`));
    stream
        .on('open', () => {stream.pipe(res)})
        .on('error', (error) => {res.end(error.toString())})
});
```

В даному коді всі шари змішані. В одному файлі знаходяться: контролер, який включає в себе інфраструктурну логіку, валідацію, бізнес-логіку. Даний підхід в розробці може існувати та працювати, але підтримувати його доволі проблематично для розробників.

2.2 Принцип архітектури серверного Веб-додатку

Архітектура веб-додатку складається з двох монументальних частин: шарів(layers) і зв'язків між ними. Потрібно спочатку декомпанувати додаток на шари, які будуть цілісними об’єктами. Наступним етапом є правильна організація ієрархії шарів і зв'язку між ними.

2.2.1 Шари

Існує класичний трирівневий підхід, який включає в себе дані, логіку та уявлення

Рисунок 2.10 - Класичний трирівневий підхід

В сучасній розробці такий підхід вважається застарілим. Проблема в тому, що основою є дані, а значить, додаток проектується в залежності від того, як дані представлені в БД, а не від того, в яких бізнес-процесах вони беруть участь.

Більш сучасний підхід передбачає, що в додатку виділено доменний шар, який працює з бізнес-логікою і є поданням реальних бізнес-процесів в коді. Однак якщо звертатися до праці Еріка Еванса «Domain-Driven Design» [8] (DDD), то виявимо там таку схему шарів програми:

Рисунок 2.11 - Схема шарів програми за працею Еріка Еванса «Domain-Driven Design»

Основою додатку, спроектованого по DDD, повинен бути домен - високорівневі політики, найважливіша логіка. Але під цим шаром лежить вся інфраструктура: шар доступу до даних (DAL), логування, моніторинг, і т. д. Тобто політики набагато більш низького рівня і меншою важливості.

Інфраструктура виявляється в центрі додатку і заміна логгеру може привести до зміни всієї бізнес-логіки.

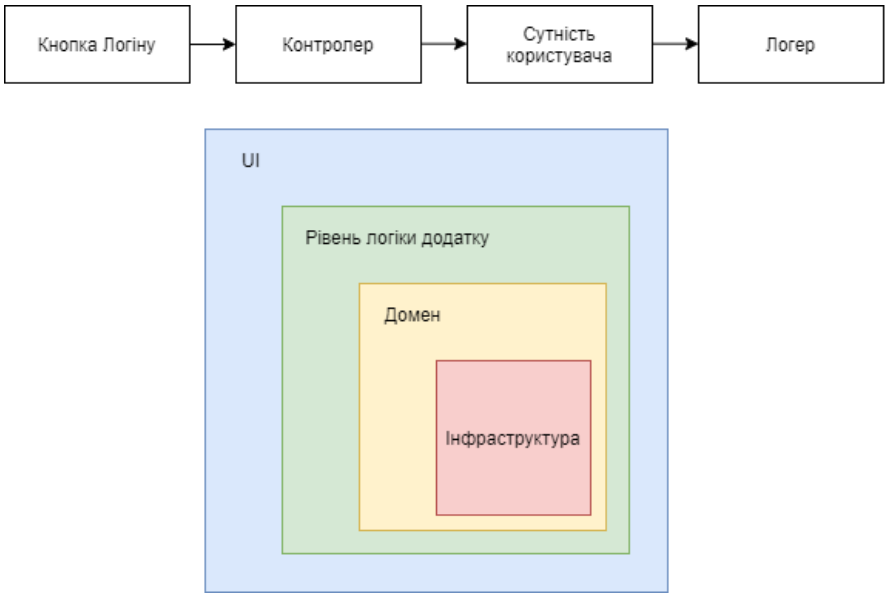

Рисунок 2.12 – Архітектура DDD

Якщо посилатися до книги Роберту Мартіну «Clean Architecture» [9], то в книзі він постулює іншу ієрархію шарів у додатках, з доменом у центрі.

Рисунок 2.13 – Архітектура за книгою Роберта Мартіна «Clean Architecture»

Отже всі чотири шари системи повинні бути розташовані за такою схемою:

Рисунок 2.14 – Схема архітектури

На цьому етапі були виділені шари та визначена їх ієрархія.

2.2.2 Зв’язки

Для того, щоб позбутися прямої залежності від інфраструктури, яка перешкоджає створенню правильної ієрархії шарів використовуються інтерфейси.

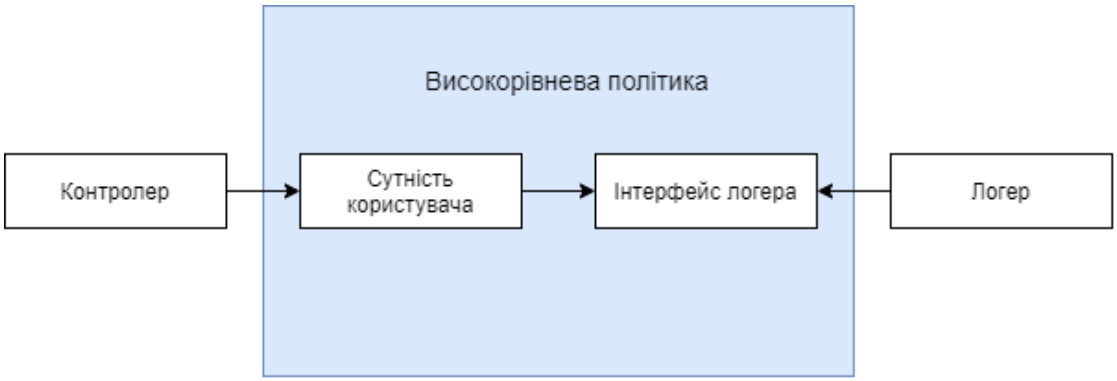

Рисунок 2.15 – Реалізація інтерфейсів

Після додавання в структуру інтерфейсів, високорівнева сутність користувача не залежить від низькорівневого логера. Також змінився тип зв’язку між ними і тепер сутність вказує контракт, який повинен бути реалізований, для підключення логеру в систему. Підключення певної реалізації логера:

```
import {Logger} from '../core/logger';
class UserEntity {
        private _logger: Logger;
        constructor() {
                this._logger = new Logger();
        }
        ...
}
...
const UserEntity = new UserEntity();
```

В даному випадку шари мають обов'язковий зв'язок. Також є залежність на файлову структуру і реалізацію. Для того, щоб видалити обов'язковий зв'язок між елементами потрібно використати інверсію залежності, яка реалізується за допомогою впровадження додаткової залежності.

```
export class UserEntity {
        constructor(private _logger: ILogger) { }
        ...
}
...
const logger = new Logger();
const UserEntity = new UserEntity(logger);
```

Після даного процесу сутність користувача не має залежності від логера. Ця сутність надає контракт і очікує, що реалізація буде відповідати цьому контракту.

З'являється проблема, яка полягає в ручній генерації екземплярів інфраструктурних сутностей. Для рішення цієї проблеми використовуються IoC-контейнери, які можуть автоматизувати цей процес.

```
export class UserEntity {
        constructor(@Inject(LOGGER) private readonly _logger: ILogger){ }
}
```

3 РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ ВЕБ -ФОРУМУ

3.1 Програмні засоби для розробки

При реалізації серверної частини веб-форуму знадобляться різні програмні додатки. Для кожної цілі при створенні додатку, потрібно обрати найбільш оптимальний варіант. На даний момент часу веб-технології розвинулись до великих масштабів і обрати із всіх аналогів стало важче із-зі їх кількості та специфікацій. Для реалізації даного бекенду були обрані наступні:

1. Node.js – це середовище виконання JavaScript. Його середовище включає в себе все, що може бути потрібно розробнику для виконання програм. Node.js перш за все призначений для створення серверних додатків на мові JavaScript. Хоча також існують проекти по написанню десктопних додатків (Electron) і навіть для створення коду для мікроконтролерів [10].
2. Express - найпопулярніший веб-фреймворк для Node. Він є базовою бібліотекою для ряду інших популярних веб-фреймворків Node. Він надає наступні механізми [11]:
    - Написання обробників для запитів з різними HTTP-методами в різних URL-адресах.
    - Інтеграція з механізмами рендеру «view», для генерації відповідей, вставляючи дані в шаблони.
    - Установка загальних параметрів веб-додатку, такі як порт для підключення і розташування шаблонів, які використовуються для відображення відповіді.
    - «Проміжне програмне забезпечення» для додаткової обробки запиту в будь-який момент в конвеєрі обробки запитів.

У той час як сам express досить мінімалістичний, розробники створили сумісні пакети проміжного програмного забезпечення для вирішення практично будь-якої проблеми з веб-розробкою. Існують бібліотеки для роботи з кукі-файлами, сеансами, входами користувачів, параметрами URL, даними POST, заголовками безпеки і багатьма іншими. Можна знайти список пакетів

проміжного програмного забезпечення, підтримуваних командою Express в Express Middleware [11].

3. MySQL - це реляційна система управління базами даних з відкритим вихідним кодом. В даний час ця СУБД одна з найбільш популярних в веб-додатках - переважна більшість CMS використовує саме MySQL (часто тільки її, без альтернатив), а майже всі веб-фреймворки підтримують MySQL вже на рівні базової конфігурації (без додаткових модулів) [12].

   З переваг СУБД MySQL варто відзначити простоту використання, гнучкість, низьку вартість володіння (щодо платних СУБД), а також масштабованість і продуктивність. Гнучкість СУБД MySQL забезпечується підтримкою великої кількості типів таблиць: користувачі можуть вибрати як таблиці типу MyISAM, що підтримують повнотекстовий пошук, так і таблиці InnoDB, що підтримують транзакції на рівні окремих записів. Є й інші типи таблиць, розроблені спільнотою.
4. Npx - інструмент, який призначений для допомоги стандартизувати досвід використання npm-пакетів: так само, як і npm спрощує установку і управління залежностями, розміщеними в реєстрі, npx спрощує використання CLI-утиліт і інших виконуваних файлів. Це значно спрощує ряд речей, які ми раніше можна було зробити за допомогою звичайного npm [13].
5. Sequelize - це ORM-бібліотека для додатків на Node.js, яка здійснює зіставлення таблиць в бд і залежностями між ними з класами. При використанні Sequelize ми можемо не писати SQL-запити, а працювати з даними як зі звичайними об'єктами. Причому Sequelize може працювати з рядом СУБД - MySQL, Postgres, MariaDB, SQLite, MS SQL Server [14].
6. Npm - це менеджер пакетів, який можна підключити в проект [15]. Наприклад, якщо розробник бажає, щоб проект використовував Bootstrap, іншому розробнику необов'язково вручну копіювати всі необхідні файли з сайту Bootstrap і зберігати цю бібліотеку в репозиторії GIT, досить вказати в конфігураційному файлі проекту одним рядком, що потрібен Bootstrap і пакетний менеджер npm встановить Bootstrap в копію проекту, при цьому

немає потрібністі зберігати всі подібні бібліотеки в репозиторії GIT вашого проекту.

7. JavaScript - це мова програмування, яка зазвичай застосовується в якості вбудованого інструменту для програмного доступу до різних об'єктів додатків. З точки зору веб-розробки, без знань цієї технології неможливо займатися створенням сучасних інтерактивних сайтів [16]. Мова JS - це те, що «оживляє» розмітку сторінок (HTML) і призначений для користувача функціонал (CMS) сайтів. За допомогою цієї мови реалізується можливість реакції сторінки або окремих її елементів на дії користувача. Сьогодні JavaScript є базовою мовою програмування для браузерів. Він повністю сумісний з операційними системами Windows, Linux, Mac OS, а також всіма популярними мобільними платформами.
8. Postman - це потужний інструмент тестування і розробки API, який при цьому має простий і інтуїтивно зрозумілий інтерфейс. Якщо ви вперше відкриваєте цю програму, знаючи в чому різниця між POST і GET, то ви без зусиль зможете відправити свій перший запит. Але є і неочевидний функціонал, який дуже полегшить процес розробки та тестування.
9. Chai - це assert бібліотека, подібна до вбудованої в Node assert. Це значно полегшує тестування, даючи багато тверджень, якими можна протистояти своєму коду [17].

3.2 Розробка бази даних

Бази даних всюди: від найпростіших блогів і директорій до надійних інформаційних систем і великих соціальних мереж. Не важливо, проста або складна база даних, наскільки істотно спроектувати її правильно. Коли база спроектована бездумно і без чіткого розуміння мети, вона не просто неефективна, але й подальша робота з базою буде потенційною проблемою для користувачів.

При проектуванні бази даних, спочатку виділимо сутності, з якими будемо працювати:

- Стаття або пост.
- Повідомлення.
- Роль користувача.
- Користувач.
- Лайки статей або постів.
- Відновлення аккаунту.

Кожна з сутностей буде представлена в вигляді таблиці з різними полями, які будуть знаходить на MySQL Server. Опишемо кожну сутність атрибутами(полями).

Стаття (topics) характеризується такими параметрами:

- Ідентифікатор(id)
- Назва(topic_name)
- Логін автора(creator_name)
- Кількість лайків за весь час(likes)
- Кількість лайків за останній тиждень(weekly_likes_counter)
- Кількість лайків за останній місяць(monthly_likes_counter)

Повідомлення (messages) характеризується такими параметрами:

- Ідентифікатор(id)
- Логін автора(author_name)
- Ідентифікатор статі(topic_id)
- Дата створення(date)
- Текст повідомлення(text)

Роль користувача (role) характеризується такими параметрами:

- Ідентифікатор(id)
- Назва ролі(role_name)

Користувач (users) характеризується такими параметрами:

- Ідентифікатор(id)
- Ім’я(first_name)
- Прізвище(second_name)

- Ідентифікатор ролі(role_id)
- Логін(login)
- Пароль(password)
- Дата народження(birthday)
- Імейл(email)

Лайки статей (topic-likes) характеризується такими параметрами:

- Ідентифікатор(id)
- Ідентифікатор статі(topic_id)
- Логін користувача(user_login)
- Дата(date)

Відновлення аккаунту (restore-keys) характеризується такими параметрами:

- Ідентифікатор(id)
- Ключ(key)
- Імейл(email)

Кожна сутність характеризується як параметрами, так і зв’язком між іншими сутностями. Для цього побудуємо ERD.

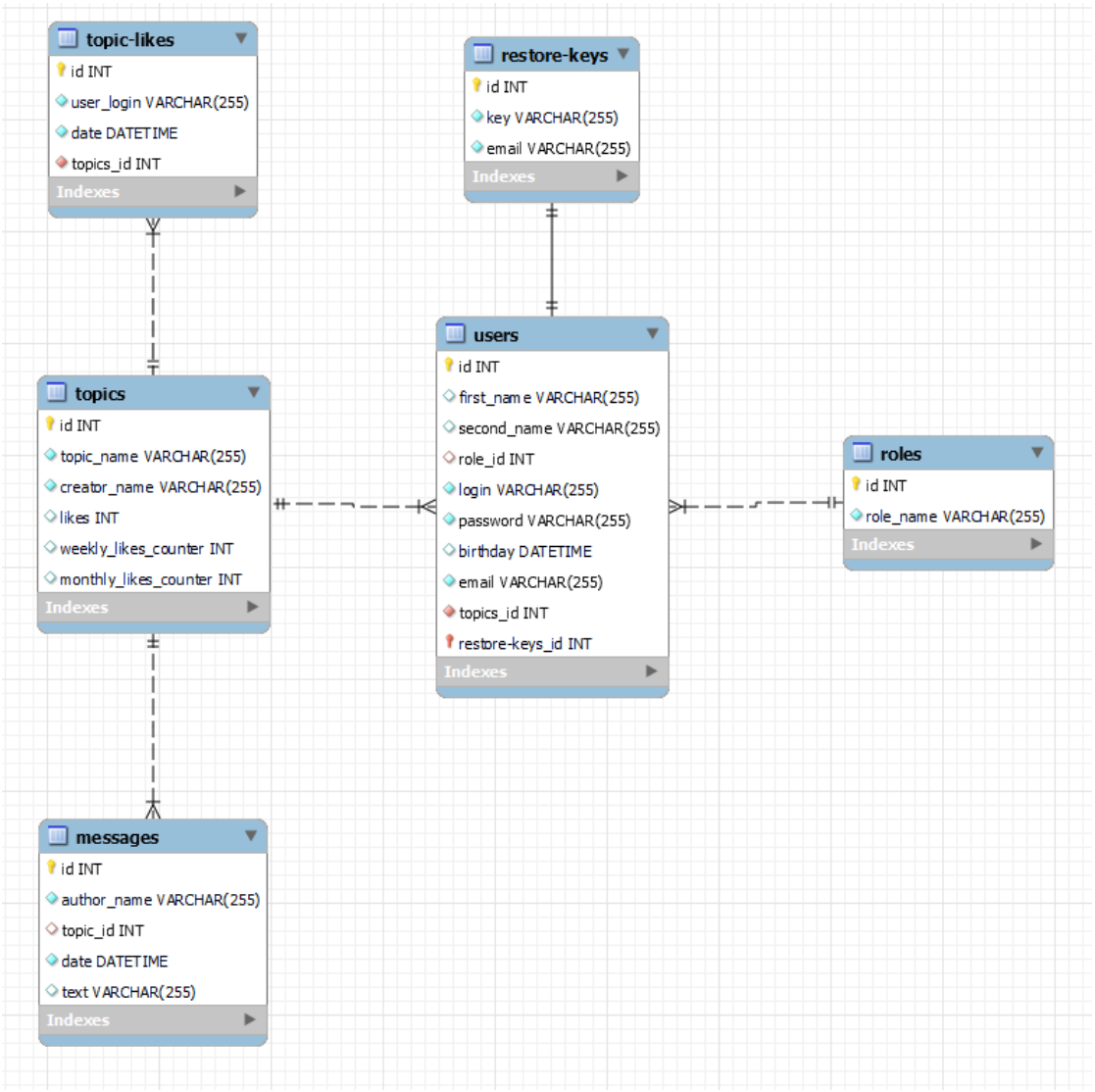

Рисунок 3.1 - ERD

При створенні ERD також були вказані типи кожної властивості, що є необхідним для створення на базі діаграми таблиць в базі даних. Так як ми не використовуємо стандартні сценарії MySQL для створення таблиць, потрібно описати їх створення за допомогою Sequelize.

Код для створення моделі повідомлень:

```
const dataConnection = require('../../user_db/data-connection.js');

const {sequelizeConnection} = dataConnection;
const sequrlizeType = sequelizeConnection.Sequelize;
const message = sequelizeConnection.define('messages', {
  id: {
    type: sequrlizeType.INTEGER,
    primaryKey: true,
    autoIncrement: true,
    allowNull: false,
  },
  author_name: {
    type: sequrlizeType.STRING,
    allowNull: false,
  },
  topic_id: {
    type: sequrlizeType.INTEGER,
    references: {
      model: 'topics',
      key: 'id',
    },
  },
  date: {
    type: sequrlizeType.DATE,
    allowNull: false,
  },
  text: {
    type: sequrlizeType.STRING,
  },
});
module.exports = message;
```

Спочатку створюємо змінні для підключення до локального серверу MySQL за семантикою Sequelize. Наступним кроком створимо об’єкт message, який і буде формувати сутність повідомлення. Після створення об’єкту експортуємо даний об’єкт.

Для роботи з сутностями в коді потрібно створити на їх основі об’єкти класів, якими будемо оперувати в реалізації бекенду. Реалізація на прикладі повідомлення має такий вигляд:

```
class Message {
  constructor(topicId, author, creationDate, text) {
    this.topicId = topicId;
    this.author = author;
    this.creationDate = creationDate;
    this.text = text;
  }
}
module.exports = Message;
```

Для початкових даних потрібно створити міграційні та спеціальні seeders файли (для прикладу використаємо об’єкт «Повідомлення»):

Лістинг міграційного файлу:

```
const messages = require('../models/message_models/message-model');

module.exports = {
  up: () => messages.sync({force: true}),
  down: (queryInterface) => queryInterface.dropTable('messages'),
};
```

Лістинг seeders файлу:

```
module.exports = {
  up: (queryInterface) => queryInterface.bulkInsert('messages', [{
    author_name: 'Darina',
    topic_id: 1,
    date: '00:00:00',
    text: 'Test Text 2',
  }], {}),

  down: (queryInterface) => queryInterface.bulkDelete('messages', null, {}),
};
```

Повний лістинг коду роботи з базами даних знаходяться в додатку А. На даному етапі переходимо до створення серверної частини додатку, який буде працювати з цими даними.

3.3 Реалізація функціоналу додатку

Для початку опишемо файлову структуру проекту.

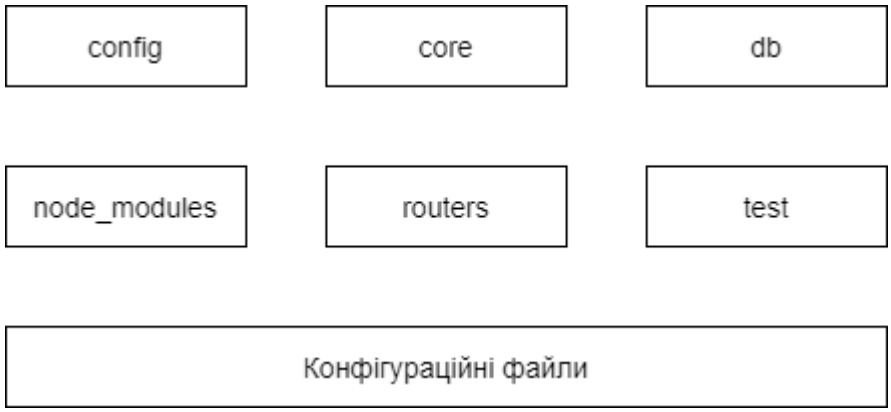

Рисунок 3.2 – Файлова структура проекту

В корені проекту знаходяться 6 папок та конфігураційні файли. Призначення кожної з папок:

- config – файл конфігурації для підключення до БД.
- core – валідація даних.
- db – файли для БД.
- node_modules – додаткові модулі, які потрібні для роботи додатку.
- routers – контролери та реалізація обробки запитів.
- test – unit тести

Також в корені проекту знаходяться такі файли:

- .env – змінні середовища..
- .sequelizerc – конфіг для Sequelize.
- main.js – основний файл, який відповідає за процес роботи серверної частини додатку.
- package.json – залежності зовнішніх пакетів.

Запуск додатку починається з виконання файлу main.js.

```
const express = require('express');
const bodyParser = require('body-parser');

const app = express();
const userFunctionalityRouter = require('./routers/user_routers/user-router.js');
const messageFunctionalityRouter = require('./routers/message_routers/message-router.js');
const messageWebSocket = require('./routers/message_routers/message-web-socket');
const accessControlRouter = require('./routers/access-control-router.js');


app.listen(3001);
app.use('/', accessControlRouter);
app.use(bodyParser.json());
app.use('/user/', userFunctionalityRouter);
app.use('/topics/', messageFunctionalityRouter);
messageWebSocket;
```

Так як ми використовуємо фреймворк Express для роботи нашого додатку, спершу нам потрібно імпортувати його модуль і створити об’єкт. Також для роботи потрібно імпортувати модулі маршрутів(routers), які будуть опрацьовувати запит користувача і перенаправляти відповідно логіки програми. Після імпорту модулів та створення об’єкту Express треба вказати через який порт буде працювати система. Для обробки HTTP запитів виконуємо таку логіку:

- Якщо запит звертається до кореню сайту, то інтерпретатор почне посилатися до accessControlRouter. Після закінчення виконання main.js продовжиться в будь-якому випадку.
- Трансформуємо тіло запиту в формат json
- Якщо URL запиту включає в себе «/user/», то йде перенаправлення до userFunctionalityRouter, який включає власну логіку роботи. Після закінчення виконання цього блоку вважаємо, що відповідь на запит користувача відправлений. Якщо URL запиту не включає в себе «/user/», то перевіряємо аналогічно на наявність «/topics/» і виконуємо аналогічні дії.

Це є принцип роботи серверного додатку, який є не настільки і складним або великим за розміром, проте вся суть архітектури полягає в тому, що кожний запит проходить через низку маршрутів по ієрархії. Якщо на сайті була б тільки одна сторінка, то все виглядало приблизно так, як описано вище, але форум це більш складна система.

Виділимо два типи функціональних вимог: функції з користувачем і функції з темами форуму.

Користувач має такі можливості:

- Авторизація.
- Реєстрація.
- Змінити дату народження.
- Змінити пароль.
- Змінити імейл.
- Відновити пароль.

Функціонал доступний для тем:

- Показати останню тему.
- Створити тему.
- Видалити тему.
- Лайкнути тему.

- Показати теми, створені певним користувачем.

За функціонал користувача відповідає userFunctionalityRouter, який був імпортований в файлі main.js. Лістинг файлу user-router.js:

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

const router = express.Router();
const signIn = require('./sign-in.js');
const signUp = require('./sign-up.js');
const signInValidator = require('../middlewares/user_middlewares/validate-sign-in-middleware.js');
const signUpValidator = require('../middlewares/user_middlewares/validate-sign-up-middleware.js');
const updateUserPersonalData = require('./change-user-personal-data.js');
const userPersonalDataValidator = require('../middlewares/user_middlewares/validate-personal-data-middleware.js');
const changePassword = require('./change-password.js');
const changeEmail = require('./change-email.js');
const authorizationMiddleware = require('../middlewares/user_middlewares/authorization-middleware.js');
const restorePassword = require('./restore-password.js');
const sendRestorePasswordCode = require('./send-restore-password-key.js');
const restoreKeyMiddleware = require('../middlewares/user_middlewares/validate-is-restore-key-does-exist-middleware');

router.use('/sign-in', signInValidator);
router.use('/sign-in', signIn);
router.use('/sign-up', signUpValidator);
router.use('/sign-up', signUp);
router.use('/profile/change-data/send', authorizationMiddleware);
router.use('/profile/change-data/send', userPersonalDataValidator);
router.use('/profile/change-data/send', updateUserPersonalData);
router.use('/profile/change-pass/send', authorizationMiddleware);
router.use('/profile/change-pass/send', changePassword);
router.use('/profile/change-email/send', authorizationMiddleware);
router.use('/profile/change-email/send', changeEmail);
router.use('/sign-in/forget-password', sendRestorePasswordCode);
router.use('/sign-in/restore-password/send-key', restoreKeyMiddleware);
router.use('/sign-in/restore-password/send-key', restorePassword);

module.exports = router;
```

Лістинг файлу message-router.js:

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

const router = express.Router();
const showTopics = require('./show-topics.js');
const deleteTopic = require('./delete-topic.js');
const validateIsAdminOrAuthor = require('../middlewares/message_middlewares/validate-is-admin-or-author-middleware.js');
const createTopic = require('./create-topic.js');
const authorizationMiddleware = require('../middlewares/user_middlewares/authorization-middleware.js');
const likeTopic = require('../message_routers/like-topic.js');
const topicLikeService = require('../controllers/message_controllers/topic-like-service.js');
const userTopics = require('../message_routers/user-topics.js');
const showTopicTop = require('./show-topic-top');

router.use('/', showTopics);
router.use('/top', showTopicTop);
router.use('/create-topic', authorizationMiddleware);
router.use('/create-topic', createTopic);
router.use('/delete-topic', authorizationMiddleware);
router.use('/delete-topic', validateIsAdminOrAuthor);
router.use('/delete-topic', deleteTopic);
router.use('/like', authorizationMiddleware);
router.use('/like', likeTopic);
router.use('/my-topics', authorizationMiddleware);
router.use('/my-topics', userTopics);

topicLikeService.setWeeklyInterval();
topicLikeService.setMonthlyInterval();

module.exports = router;
```

Принцип обробки запиту аналогічно файлу main.js. Після виклику функції use у об’єкта router перевіряється відповідність URL-адреси в запиті до відповідної функції.

Для валідації вхідних даних використовуються відповідні роутери для перевірки коректності даних для їх подальшої обробки.

Лістинг signInValidator:

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

const router = express.Router();
const LoginRequest = require('../../../db/db_objects/user_db_objects/login-request.js');
const signInValidation = require('../../../core/validations/user_validation/sign-in-validation.js');

router.post('/', (request, response, next) => {
  const loginRequestData = new LoginRequest(request.body.login, request.body.password);
  if (signInValidation.validateSignIn(loginRequestData, response)) {
    return next();
  }
  response.status(400).send('Данные введены неправильно');
});
module.exports = router;
```

Для валідації використовується зовнішня функція, в якій описано алгоритм валідації даних:

```
const mailValidation = require('./email-validation.js');
const passValidation = require('./password-validation.js');
const loginValidation = require('./login-validation.js');

function validateSignIn(loginRequest) {
  return !!((mailValidation.validateEmail(loginRequest.login)
    || loginValidation.validateLogin(loginRequest.login))
    && passValidation.validatePassword(loginRequest.password));
}
module.exports.validateSignIn = validateSignIn;
```

В даному випадку, якщо користувач ввів не правильні або не валідні дані, то він отримає відповідь з сервера, в якій буде описана помилка. Якщо валідація даних пройшла успішно, то запит починає опрацьовуватись роутером signIn:

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

const router = express.Router();
const LoginRequest = require('../../db/db_objects/user_db_objects/login-request.js');
const authenticationService = require('../controllers/user_controllers/authentification-service.js');

router.post('/', (request, response) => {
  const loginRequestData = new LoginRequest(request.body.login, request.body.password);
  authenticationService.signIn(loginRequestData, response);
});
module.exports = router;
```

На цьому етапі до бази даних відправляється запит на авторизацію і автентифікація користувача в системі

За аналогією поступового опрацювання запиту для авторизації було реалізовано весь функціонал. Повний лістинг коду знаходиться в додатку Б.

3.4 Тестування серверної частини

Для перевірки працездатності системи потрібно провести його тестування. Кількість способів протестувати працездатність бекенду велика. Для нашого додатку будемо використовувати ручне тестування за допомогою Postman. Також всесвітньою практикою в веб-розробці є написання модульних тестів, які допомагають розробникам підтримувати якість коду і відсутність помилок при зміні або видаленні певного модуля з системи. Модульні тести є автоматичними.

3.4.1 Перевірка працездатності додатку

Для того, щоб перевірити додаток на працездатність потрібно створити конфігураційні файли, встановити модулі Node.js і запустити сервер.

Файл .env

```
DB_PASS=gfhjkm1212
DB_USER=root
DB_NAME=project
JWT_KEY=test
EMAIL_NAME=Project email
EMAIL_PASSWORD=Project email password
```

Рисунок 3.3 – Конфігурація .env

Файл .sequelizerc

```
var path = require('path')

module.exports = {
  'config': path.resolve('config', 'config.js'),
  'migrations-path': path.resolve('db', 'migrations'),
  'seeders-path':    path.resolve('db', 'seeders'),
}
```

Рисунок 3.4 – Конфігурація .sequelizerc

Файл config.js

```
require('dotenv').config();

const connectionConfig = {
  database: process.env.DB_NAME,
  username: process.env.DB_USER,
  password: process.env.DB_PASS,
  host: 'localhost',
  dialect: 'mysql',
  logging: false,
  define: {
    timestamps: false,
  },
};

module.exports = connectionConfig;
```

Рисунок 3.5 – Конфігурація config.js

Для інсталяції потрібних модулів потрібно виконати команду «npm install».

Потрібно виконати міграцію даних в MySQL Server. Виконаємо дві команди «npx sequelize-cli db:migrate» і «npx sequelize-cli db:seed:all».

Сервер готовий до запуску. Запускаємо командою «node .\main.js». Після запуску повинен створитись процес в операційній системі.

3.4.2 Тестування за допомогою Postman

Перед початком тестування виведемо інформацію, яка існує на даний момент в таблиці users запитом SQL:

SELECT * FROM project.users;

Відповідь:

id	first_name	second_name	role_id	login	password	birthday	email
1	NULL	NULL	1	Nita	$2b$10$kjzonDTMA1lwe5/592kpdej4i4MOBG5G...	NULL	Nta@ad.erd
2	NULL	NULL	1	Darina	$2b$10$f9Rgn89kLYroydMHiYSnIOdczN6K0w03...	NULL	Darina@ad.erd
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 3.6 – таблиця користувачів

Протестуємо реалізацію реєстрації нашого додатку. Для цього напишемо POST запит на локальний сервер, порт 3001. URL запиту виглядає наступним чином:

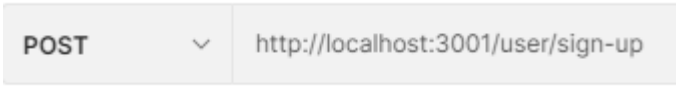

Рисунок 3.7 – URL запиту реєстрації

В тілі запиту потрібно передати логін, імейл та пароль в форматі JSON:

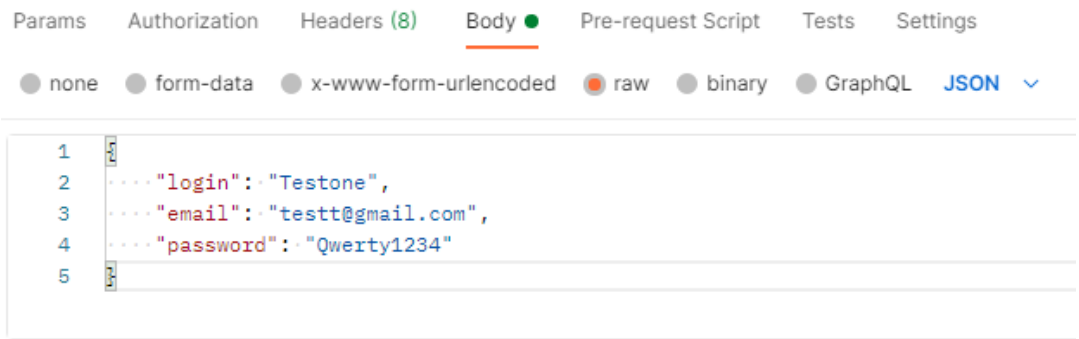

Рисунок 3.8 – Тіло запиту реєстрації

Відправляємо запит на сервер і отримаємо повідомлення про успішну реєстрацію.

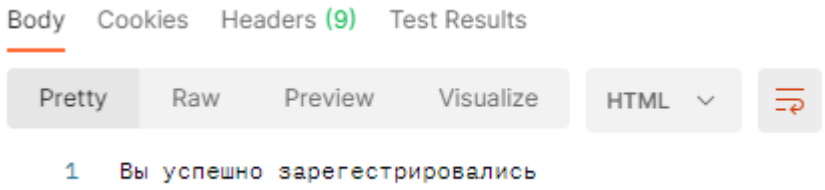

Рисунок 3.9 – Успішна реєстрація

Наступним буде перевірятись спроможність авторизації користувача. URL запиту має вигляд «http://localhost:3001/user/sign-in». Це також є POST запитом.

В тілі запиту передаємо логін та пароль в форматі JSON. Відправляємо запит на сервер і отримаємо токен та масив лайків. Код запиту 200, що означає про успішність обробки запиту.

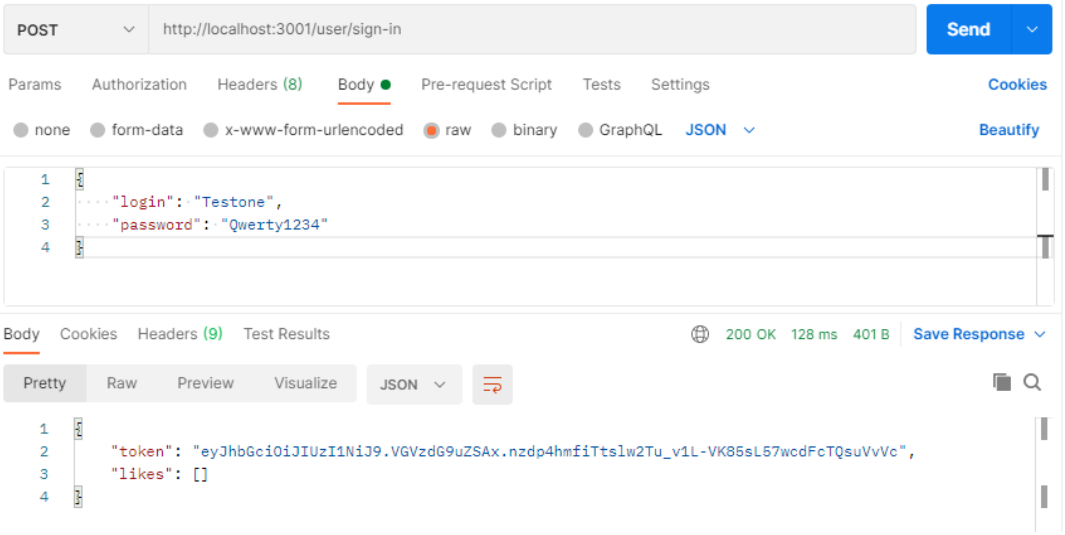

Рисунок 3.10 – Запит авторизації.

Для перевірки отримання всіх тем будемо використовувати GET запит з URL виду «http://localhost:3001/topics/1» без параметрів тіла запиту.

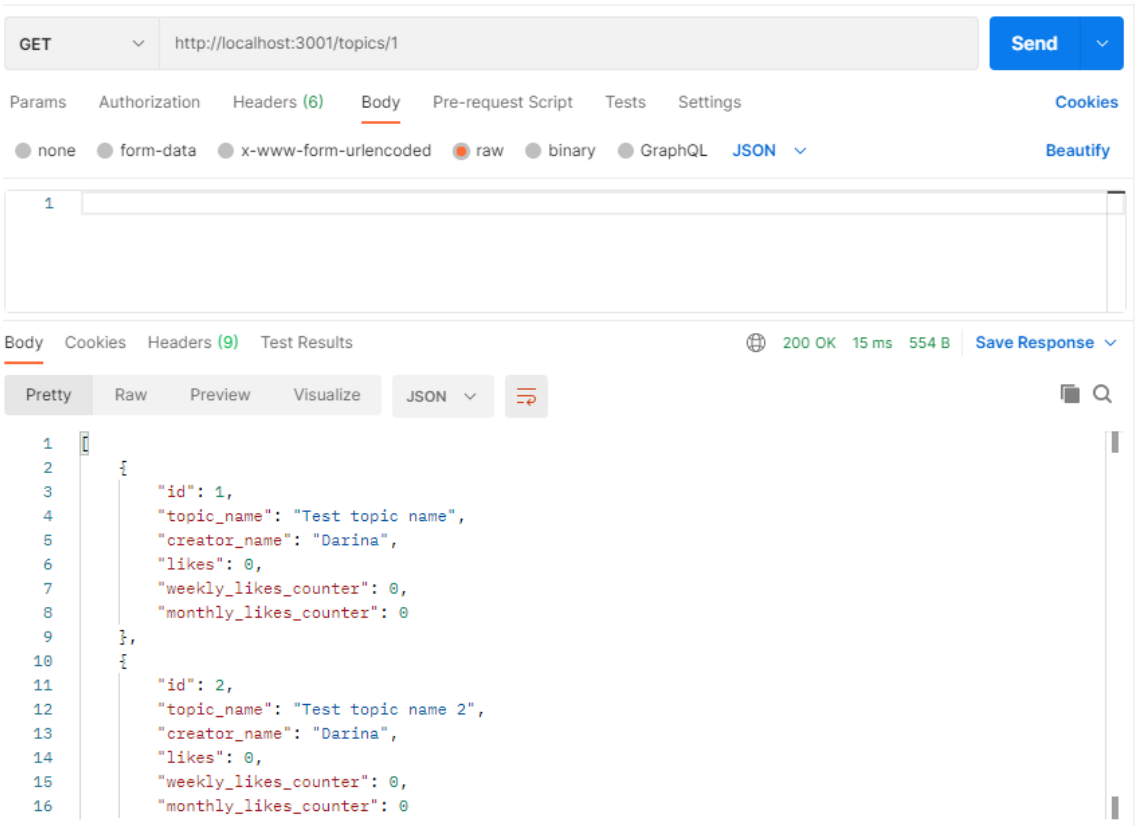

Рисунок 3.11 – Запит всіх тем

Код запиту 200, в тілі запиту ми отримали дані про всі наявні теми в форматі JSON для подальшої роботи з ними на клієнтській частині веб-додатку.

Для зміни персональної інформації користувача URL запиту методом POST буде «http://localhost:3001/user/profile/change-data/send». В тілі запиту потрібно вказати логін, ім’я, прізвище та дату народження. В headers потрібно передати токен авторизації.

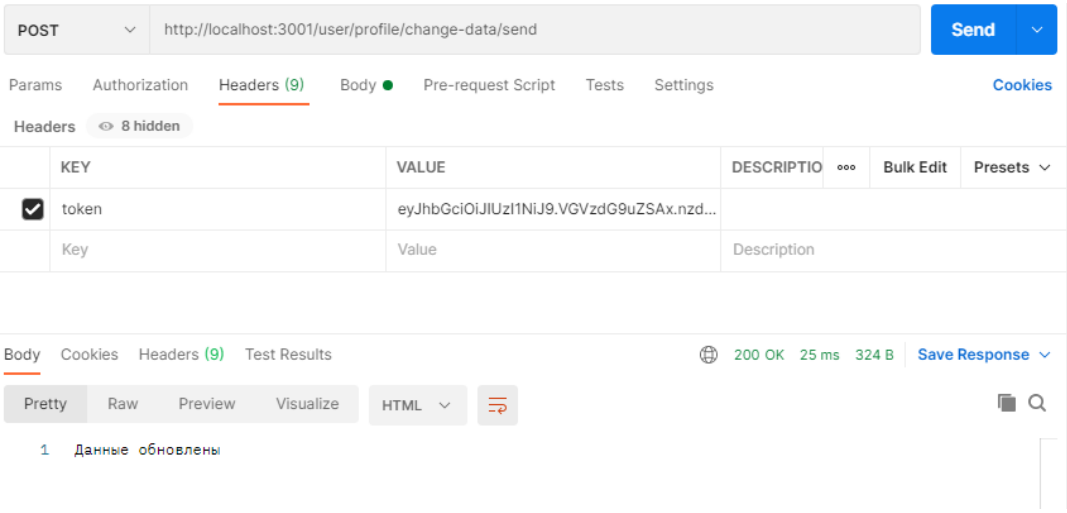

Рисунок 3.12 – Запит зміни персональних даних.

Дані успішно змінилися.

Важливим елементом функціоналу користувача є можливість зміни паролю. URL виглядає таким чином «http://localhost:3001/user/profile/change-pass/send». Це POST запит який може відправляти тільки авторизований користувач і для цього в запиті потрібно передати токен авторизації.

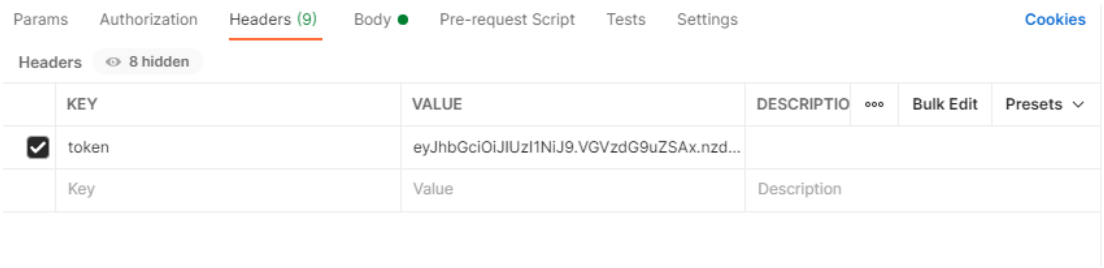

Рисунок 3.13 – Токен авторизації

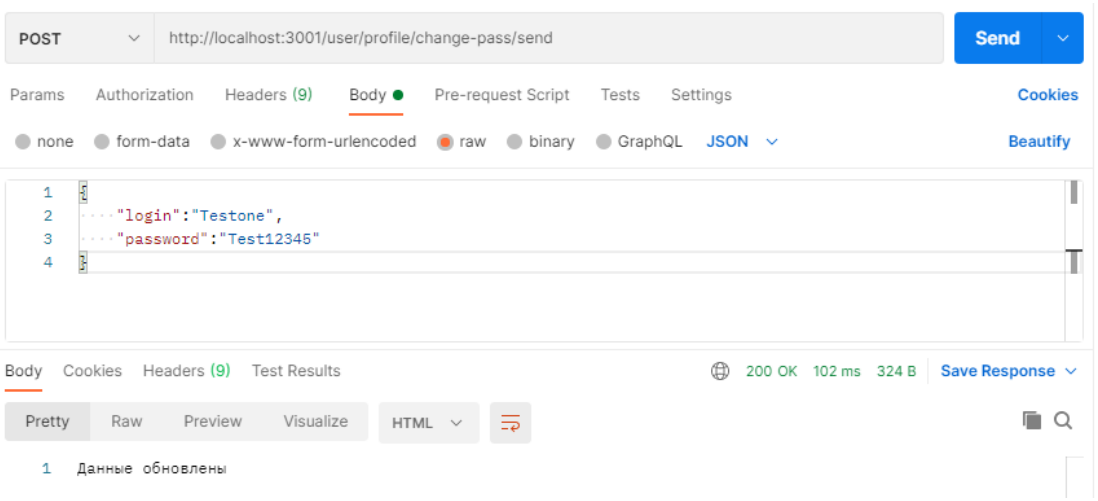

Рисунок 3.14 – Запит зміни паролю

Реалізований функціонал працює, відповідно вимогам і можна стверджувати, що тестування додатку успішне.

3.4.3 Unit тестування

Наступним видом тестування додатку є Unit тестування. Unit тестування - це вид тестування програмного забезпечення, де тестуються окремі блоки або компоненти програмного забезпечення. Метою є перевірити, що кожна одиниця програмного коду працює належним чином [18].

В даному тестуванні будемо перевіряти такі функціональні можливості:

- Реєстрація користувачів.
- Авторизація користувачів.

- Зміна персональних даних.

При тестуванні використовувались як коректні, так і не правильні вхідні дані для більш детальної перевірки.

Початкові налаштування файлу з тестами test,js

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

const app = express();
const bodyParser = require('body-parser');

const chaiHttp = require('chai-http');

const chai = require('chai');
const userModel = require('../db/models/user_models/user-model.js');
const user = require('../routers/user_routers/user-router');

const {expect} = chai;

chai.use(chaiHttp);
app.use(bodyParser.json());
```

3.4.3.1 Тестування авторизації з коректними даними

```
describe('user', () => {
  describe('sign-in', () => {
    describe('Correct', () => {
      it('Sign In correct request with email and password', (done) => {
        const request = {login: 'Darina@ad.erd', password: 'A1267ddc'};
        chai
            .request(app.use('/', user))
            .post('/sign-in')
            .send(request)
            .end((err, res) => {
              expect(res).to.have.status(200);
              done();
            });
      });
      it('Sign In correct request with login and password', (done) => {
        const request = {login: 'Darina', password: 'A1267ddc'};
        chai
            .request(app.use('/', user))
            .post('/sign-in')
            .send(request)
            .end((err, res) => {
              expect(res).to.have.status(200);
              done();
            });
      });
    });
```

3.4.3.2 Тестування авторизації з некоректними даними

```
describe('Incorrect', () => {
  it('Sign in with no password', (done) => {
    const request = {login: 'Darina', password: ''};
    chai
        .request(app.use('/', user))
        .post('/sign-in')
        .send(request)
        .end((err, res) => {
          expect(res).to.have.status(400);
          done();
        });
  });
  it('Sign in with no number at password', (done) => {
    const request = {login: 'Darina@ad.erd', password: 'Password'};
    chai
        .request(app.use('/', user))
        .post('/sign-in')
        .send(request)
        .end((err, res) => {
          expect(res).to.have.status(400);
          done();
        });
  });
```

Інші варіації тестів знаходяться в додатку Б.

3.4.3.3 Тестування реєстрації з коректними даними

```
describe('sign-Up', () => {
  before(() => {
    userModel.user.destroy({
      where: {
        login: 'Nita',
      },
    });
  });
  after(() => {
    userModel.user.destroy({
      where: {
        login: 'Nita',
      },
    });
  });
  describe('Correct', () => {
    it('Sign up wit correct email,login,password', (done) => {
      const request = {login: 'Nita', email: 'Nta@ad.erd', password: 'Password123'};
      chai
          .request(app.use('/', user))
          .post('/sign-Up')
          .send(request)
          .end((err, res) => {
            expect(res).to.have.status(200);
            done();
          });
    });
  });
```

3.4.3.4 Тестування реєстрації з некоректними даними

```
describe('Incorrect', () => {
  it('Sign up with no password', (done) => {
    const request = {login: 'Darina', email: 'Niita@ad.erd', password: ''};
    chai
        .request(app.use('/', user))
        .post('/sign-Up')
        .send(request)
        .end((err, res) => {
          expect(res).to.have.status(400);
          done();
        });
  });
  it('Sign up with no number at password', (done) => {
    const request = {login: 'Darina', email: 'Niita@ad.erd', password: 'Password'};
    chai
        .request(app.use('/', user))
        .post('/sign-Up')
        .send(request)
        .end((err, res) => {
          expect(res).to.have.status(400);
          done();
        });
  });
```

3.4.3.5 Тестування коректної зміни персональних даних користувача

```
describe('change-data', () => {
  describe('Change additional data', () => {
    describe('Correct change additional data', () => {
      it('Correct change additional data', (done) => {
        const request = {
          login: 'Nita',
          first_name: 'Nikita',
          second_name: 'Stovpak-Karina',
          birthday: '2001-01-10',
        };
        chai
            .request(app.use('/', user))
            .post('/profile/change-data/send')
            .send(request)
            .end((err, res) => {
              expect(res).to.have.status(200);
              done();
            });
      });
    });
```

3.4.3.6 Тестування некоректної зміни персональних даних користувача

```
describe('Incorrect change additional data', () => {
  it('Incorrect date', (done) => {
    const request = {
      login: 'Nita',
      first_name: 'Nikita',
      second_name: 'Stovpak-Karina',
      birthday: '2001:01:10',
    };
    chai
        .request(app.use('/', user))
        .post('/profile/change-data/send')
        .send(request)
        .end((err, res) => {
          expect(res).to.have.status(400);
          done();
        });
  });
  it('Incorrect name', (done) => {
    const request = {
      login: 'Nita',
      first_name: 'Nikita-',
      second_name: 'Stovpak-Karina',
      birthday: '2001-01-10',
    };
    chai
        .request(app.use('/', user))
        .post('/profile/change-data/send')
        .send(request)
        .end((err, res) => {
          expect(res).to.have.status(400);
          done();
        });
  });
```

Результати тестів

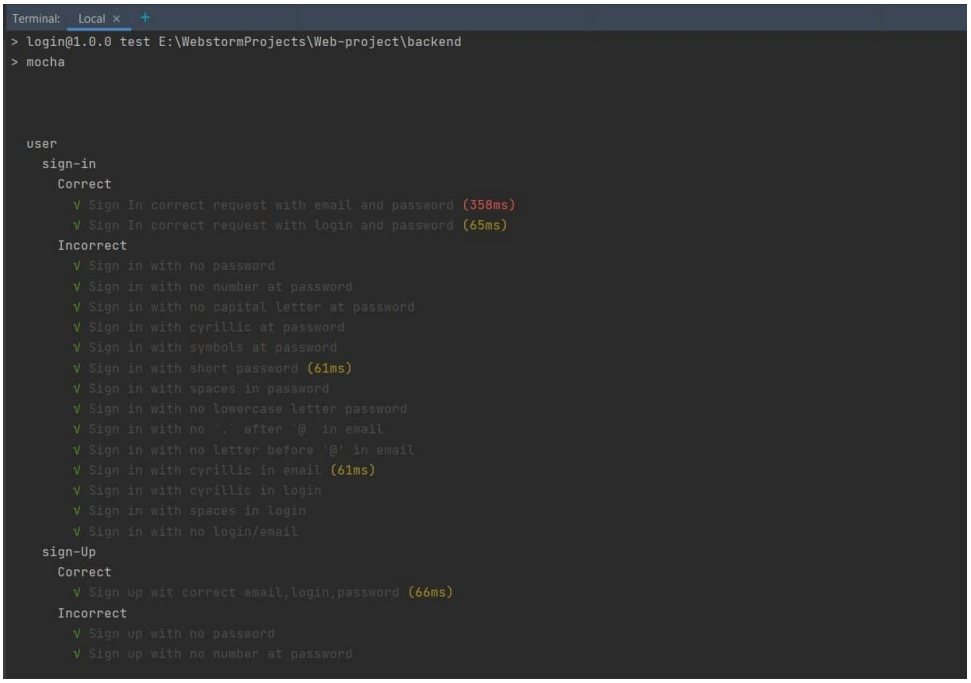

Рисунок 3.15 – Результати тестування

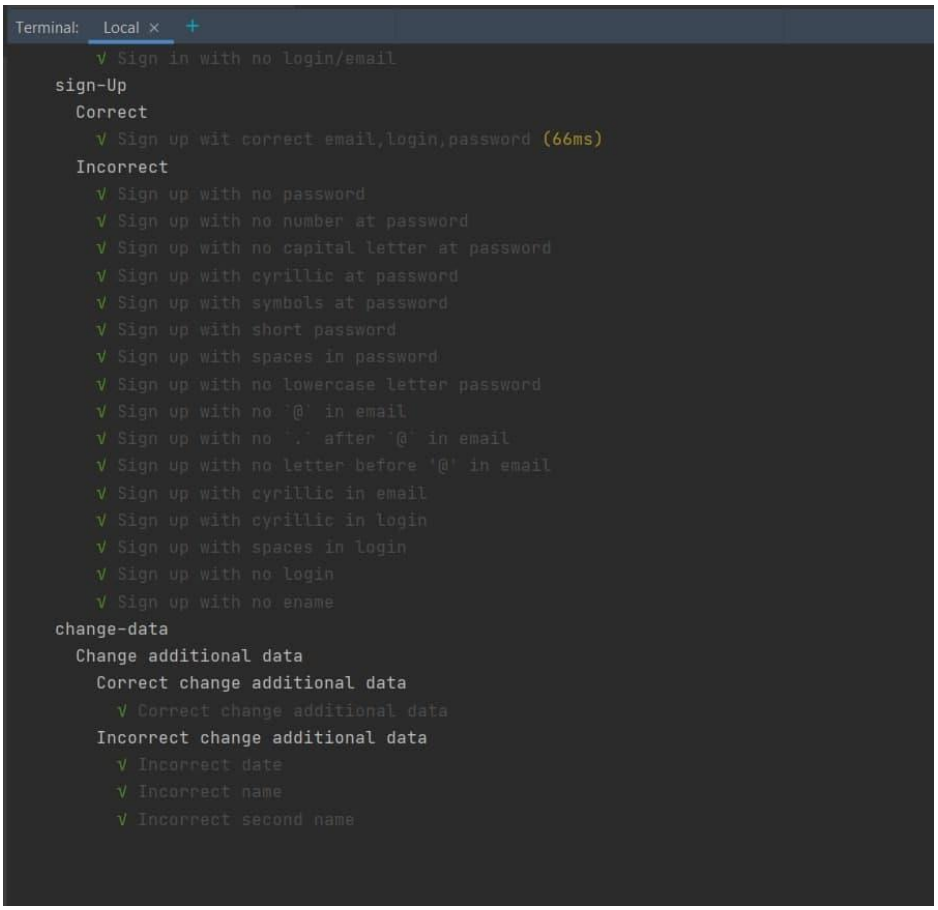

Рисунок 3.16 - Результати тестування

Всі тести пройшли успішно, з чого можна зробити висновок про коректну роботу серверної частини веб-додатку.

ВИСНОВКИ

Дана робота була направлена на створення серверної частини веб-форуму на автомобільну тематику, який є досі актуальною темою в сучасному діджиталізованому світі. Більшість сайтів-аналогів приділяють мало уваги на серверну частину додатку, сконцентрувавшись на зовнішності, що є помилкою, опираючись на аналіз конкурентних платформ. Правильний підхід при проектуванні бази даних та серверної архітектури є обов’язковою вимогою для створення веб-додатку з певними функціональними можливостями.

У випускній роботі була створена серверна частина веб-форуму з використанням реляційної бази даних, мови програмування JavaScript в симбіозі з фреймворками. Дана система була протестована і показала свою працездатність, що свідчить про готовність до міграції на хостинг сервери для запуску платформи веб-форуму в інтернет. В подальшому планується розвивати веб-форум та популяризувати його, через що прийдеться допрацьовувати як готовий функціонал для дуже великих об’ємів трафіку та імплементації нового функціоналу.

СПИСОК ЛІТЕРАТУРИ

1. https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1%D1%84%D0%BE%D1%80%D1%83%D0%BC
2. http://forums.cambridgesoft.com/help/english/What_is_a_forum.htm
3. https://hackernoon.com/an-introduction-to-backend-architecture-541g3zze
4. https://vitejs.dev/guide/ssr.html#pre-rendering-ssg
5. https://ru.reactjs.org/docs/jsx-in-depth.html
6. https://bartwullems.blogspot.com/2017/01/mini-spassingle-page-apps.html
7. https://medium.com/frontend-at-scale/frontend-architectural-patterns-backend-for-frontend-29679aba886c
8. Evans E. Domain-Driven Design - Tackling Complexity in the Heart of Software. — Addison-Wesley, 2004. — 529 с. — ISBN 978-0-3211-2521-7
9. Clean Architecture: A Craftsman's Guide to Software Structure and Design (англ.). — Prentice Hall, 2017.
10. Янг А., Мек Б., Кантелон М. Node.js в дії. - 2-е вид .. - СПб .: «Пітер», 2018. - С. 432. - ISBN 978-5-496-03212-4.
11. https://expressjs.com/
12. https://mchost.ru/articles/chto-takoe-mysql/
13. https://habr.com/ru/company/ruvds/blog/423705/
14. https://sequelize.org/
15. https://www.hostinger.com.ua/rukovodstva/chto-takoe-npm
16. https://developer.mozilla.org/ru/docs/Learn/JavaScript/First_steps/What_is_JavaScript
17. https://www.npmjs.com/package/chai
18. https://habr.com/ru/post/169381/

ДОДАТОК А

core\validations\message_validation\is-author-validation.js
```
function isAuthor(userLogin, authorLogin) {
  return userLogin === authorLogin;
}
module.exports = isAuthor;
```
core\validations\user_validation\date-validator.js
```
function validateDate(date) {
  const regularExpression = /([0-9]{4}-(0[1-9]|1[012])-(0[1-9]|1[0-9]|2[0-9]|3[01]))|(0000-00-00)/;
  return date.match(regularExpression) !== null;
}
module.exports = validateDate;
```
core\validations\user_validation\email-validation.js
```
const emailValidator = require('email-validator');
function validateEmail(email) {
  return emailValidator.validate(email);
}
module.exports.validateEmail = validateEmail;
```
core\validations\user_validation\is-admin-validation.js
```
function isAdmin(userRole) {
  return userRole === 1;
}
module.exports = isAdmin;
```
core\validations\user_validation\is-restore-key-exist.js
```
const restorePasswordKeyModel = require('../../../db/models/user_models/restore-password-key-model.js');
function validateRestoreKeyDoesExist(email) {
  return restorePasswordKeyModel.findOne({
    raw: true,
    where: {
      email: email,
    },
  });
}
module.exports = validateRestoreKeyDoesExist;
```
core\validations\user_validation\login-validation.js
```
function validateLogin(login) {
  return login.match(/[^a-z0-9]/gi) === null && login.length !== 0;
}

module.exports.validateLogin = validateLogin;
```
core\validations\user_validation\name-validation.js
```
function validateName(name) {
  const regularExpression = /^([A-Z][a-z]+-[A-Z][a-z]+)$|^([A-Z][a-z]+)$/;
  return name.match(regularExpression) !== null || name.length === 0;
}
module.exports = validateName;
```
core\validations\user_validation\password-validation.js
```
const PasswordValidator = require('password-validator');
const schema = new PasswordValidator();
schema
    .is().min(8)
    .is().max(100)
    .has()
    .uppercase()
    .has()
    .lowercase()
    .has()
    .digits()
    .has()
    .not()
    .spaces()
    .has()
    .not()
    .symbols();
function validatePassword(password) {
  return !!(password.match(/([^a-z0-9])/gi) === null && schema.validate(password));
}
```

```
module.exports.validatePassword = validatePassword;
core\validations\user_validation\sign-in-validation.js
const mailValidation = require('./email-validation.js');
const passValidation = require('./password-validation.js');
const loginValidation = require('./login-validation.js');
function validateSignIn(loginRequest) {
  return !!((mailValidation.validateEmail(loginRequest.login)
    || loginValidation.validateLogin(loginRequest.login))
    && passValidation.validatePassword(loginRequest.password));
}
module.exports.validateSignIn = validateSignIn;
core\validations\user_validation\sign-up-validation.js
const emailValidation = require('./email-validation.js');
const passValidation = require('./password-validation.js');
const loginValidation = require('./login-validation.js');
function validateSignUp(registrationRequest) {
  return !!(emailValidation.validateEmail(registrationRequest.email)
    && passValidation.validatePassword(registrationRequest.password)
    && loginValidation.validateLogin(registrationRequest.login));
}
module.exports.validateSignUp = validateSignUp;
core\validations\user_validation\user-personal-data-validation.js
const nameValidation = require('./name-validation.js');
const dateValidator = require('./date-validator.js');
function validatePersonalData(userPersonalData) {
  return !!(nameValidation(userPersonalData.firstName)
    && nameValidation(userPersonalData.secondName)
    && dateValidator(userPersonalData.birthday));
}
module.exports = validatePersonalData;
routers\access-control-router.js
const express = require('express');
const router = express.Router();
router.use(function(req, res, next) {
  res.header('Access-Control-Allow-Origin', '*'); // update to match the domain you will make the request from
  res.header('Access-Control-Allow-Headers', '*');
  next();
});

module.exports = router;
routers\controllers\message_controllers\message-service.js
const messageModel = require('../../../db/models/message_models/message-model.js');
function createMessage(message) {
  messageModel.create({
    author_name: message.author_name,
    topic_id: message.topicId,
    date: message.creationDate,
    text: message.text,
  });
}
function showOldMessages(topicId) {
  return messageModel.findAll({
    raw: true,
    where: {
      topic_id: topicId,
    },
  });
}
function deleteMessage(messageToDeleteId) {
  messageModel.destroy({
    raw: true,
    where: {
      id: messageToDeleteId,
    },
  });
}
function updateMessage(messageToUpdateId, messageText) {
  messageModel.update({text: messageText}, {
    raw: true,
    where: {
      id: messageToUpdateId,
    },
  });
}
```

```
function findMessage(messageId) {
  return messageModel.findOne({
    raw: true,
    where: {
      id: messageId,
    },
  });
}
module.exports.createMessage = createMessage;
module.exports.showOldMessages = showOldMessages;
module.exports.deleteMessage = deleteMessage;
module.exports.updateMessage = updateMessage;
module.exports.findMessage = findMessage;
routers\controllers\message_controllers\topic-like-service.js
const topicLikeModel = require('../../../db/models/message_models/topic-likes-model.js');
const topicModel = require('../../../db/models/message_models/topic-model.js');
const topicService = require('./topic-service.js');
const sequelize = require('sequelize');
const sequelizeOperators = sequelize.Op;
const moment = require('moment');
function isFirstDayOfTheMonth() {
  const date = new Date();
  return (date.getDate() === 1);
}
function doesLikesExist(Likes) {
  return !!Likes[0];
}
function findLikes(userLogin) {
  return topicLikeModel.findAll({
    where: {
      user_login: userLogin,
    },
  });
}

function findMonthlyOrWeeklyTopicLikes(searchingType, searchingValue, topics, i) {
  return topicLikeModel.findAll({
    raw: true,
    where: {
      topic_id: topics[i].id,
      date: {
        [sequelizeOperators.gte]: moment().subtract(searchingValue, searchingType).toDate(),
      },
    },
  });
}
function updateTopTopicLikes(Likes, topicId, type) {
  if (doesLikesExist(Likes)) {
    topicModel.update({[type]: Likes.length}, {
      where: {
        id: topicId,
      },
    });
  } else {
    topicModel.update({[type]: 0}, {
      where: {
        id: topicId,
      },
    });
  }
}
```

```
function countWeeklyLikes() {
  const type = 'weekly_likes_counter';
  topicService.findAllTopics().then((topics) => {
    for (let i=0; i<topics.length; i++) {
      const topicId = topics[i].id;
      findMonthlyOrWeeklyTopicLikes('days', 7, topics, i).then((Likes)=>{
        updateTopTopicLikes(Likes, topicId, type);
      });
    }
  });
}
function countMonthlyLikes() {
  if (isFirstDayOfTheMonth()) {
    const type = 'monthly_likes_counter';
    topicService.findAllTopics().then((topics) => {
      for (let i=0; i<topics.length; i++) {
        const topicId = topics[i].id;
        findMonthlyOrWeeklyTopicLikes('months', 1, topics, i).then((Likes)=>{
          updateTopTopicLikes(Likes, topicId, type);
        });
      }
    });
  } else {
    return;
  }
}
function likeTopic(topicId, userLogin) {
  topicLikeModel.findOne({
    raw: true,
    where: {
      topic_id: topicId,
      user_login: userLogin,
    },
  }).then((topicLike) => {
    if (!topicLike) {
      const date = new Date();
      topicLikeModel.create({
        topic_id: topicId,
        user_login: userLogin,

        date: date,
      });
      topicModel.increment('likes', {
        by: +1,
        where: {
          id: topicId,
        },
      });
    } else {
      topicLikeModel.destroy({
        where: {
          topic_id: topicId,
          user_login: userLogin,
        },
      });
      topicModel.increment('likes', {
        by: -1,
        where: {
          id: topicId,
        },
      });
    }
  });
}
function countLikes(topicId) {
  return topicModel.findOne({
    raw: true,
    where: {
      topic_id: topicId,
    },
  });
}
```

```
function deleteLikes(topicId) {
  topicLikeModel.destroy({
    raw: true,
    where: {
      topic_id: topicId,
    },
  });
}
function setWeeklyInterval() {
  setInterval(
      () => countWeeklyLikes(),
      604800000,
  );
}
function setMonthlyInterval() {
  setInterval(
      () => countMonthlyLikes(),
      86400000,
  );
}
module.exports.setMonthlyInterval = setMonthlyInterval;
module.exports.setWeeklyInterval = setWeeklyInterval;
module.exports.countLikes = countLikes;
module.exports.findLikes = findLikes;
module.exports.deleteLikes = deleteLikes;
module.exports.likeTopic = likeTopic;
routers\controllers\message_controllers\topic-service.js
const topicModel = require('../../../db/models/message_models/topic-model.js');
const messageModel = require('../../../db/models/message_models/message-model');
function createTopic(topic, response) {
  topicModel.create({
    topic_name: topic.name,
    creator_name: topic.creator,
  });
  response.status(200);
}

function findAllTopics() {
  return topicModel.findAll({
    raw: true,
  });
}
function findTopic(topicId) {
  return topicModel.findOne({
    raw: true,
    where: {
      id: topicId,
    },
  });
}
function deleteTopic(topicToDeleteId) {
  topicModel.destroy({
    raw: true,
    where: {
      id: topicToDeleteId,
    },
  });
}
function clearingTopic(topicId) {
  messageModel.destroy({
    raw: true,
    where: {
      topic_id: topicId,
    },
  });
}
```

```
function findUserTopics(userLogin) {
  return topicModel.findAll({
    raw: true,
    where: {
      creator_name: userLogin,
    },
  });
}
function findTopicsOnPage(page) {
  const limit = 10;
  const offset = (page-1) * limit;
  return topicModel.findAll({
    offset,
    limit,
    raw: true,
  });
}
function findTopTopics(type) {
  const limit = 10;
  return topicModel.findAll({
    limit,
    order: [[
      type, 'DESC'],
    ],
  });
}

module.exports.findTopTopics = findTopTopics;
module.exports.findTopicsOnPage = findTopicsOnPage;
module.exports.findUserTopics = findUserTopics;
module.exports.createTopic = createTopic;
module.exports.findAllTopics = findAllTopics;
module.exports.findTopic = findTopic;
module.exports.deleteTopic = deleteTopic;
module.exports.clearingTopic = clearingTopic;
routers\controllers\user_controllers\authentification-service.js
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const userService = require('./user-service.js');
const topicLikeService = require('../message_controllers/topic-like-service.js');
require('dotenv').config();

function signUp(request, response) {
  userService.exists(request.login, request.email)
      .then((User) => {
        if (!User) {
          userService.createAccount(request, response);
          return;
        }
        response.status(409).send('Такой пользователь уже существует');
      }).catch((err) => response.status(500).send(err));
}
```

```
function signIn(request, response) {
  userService.exists(request.login, request.login)
      .then((User) => {
        if (!User) {
          response.status(400).send('Данные введены неправильно');
        } else {
          bcrypt.compare(request.password, User.password, (err, res) => {
            if (res) {
              const responseStr = `${User.login} ${User.role_id.toString()}`;
              const token = jwt.sign(responseStr, process.env.JWT_KEY);
              topicLikeService.findLikes(User.login).then((likes) =>{
                response.status(200).json({token: token, likes: likes});
              });
            } else {
              response.status(400).send('Данные введены неправильно');
            }
          });
        }
      }).catch((err) => response.status(500).send(err));
}
module.exports.signUp = signUp;
module.exports.signIn = signIn;
routers\controllers\user_controllers\authorization-service.js
const jwt = require('jsonwebtoken');
function checkAuthorization(autHeader) {
  if (!autHeader) {
    return false;
  }
  try {
    jwt.verify(autHeader, process.env.JWT_KEY);
  } catch (e) {
    if (e instanceof jwt.JsonWebTokenError) {
      return false;
    }
  }
  return true;
}

module.exports = checkAuthorization;
routers\controllers\user_controllers\jwt-service.js
const jwt = require('jsonwebtoken');
function getLogin(autHeader) {
  const token = jwt.decode(autHeader);
  const login = token.replace(/\s[0-1]$/, '');
  return login;
}
function getRole(autHeader) {
  const token = jwt.decode(autHeader);
  const role = token.replace(/\S+\s/, '');
  return role;
}
module.exports.getLogin = getLogin;
module.exports.getRole = getRole;
routers\controllers\user_controllers\user-service.js
const bcrypt = require('bcrypt');
const sequelize = require('sequelize');
const userModel = require('../../../db/models/user_models/user-model.js');
const passwordValidator = require('../../../core/validations/user_validation/password-validation.js');
const emailValidation = require('../../../core/validations/user_validation/email-validation.js');
const jwtService = require('../user_controllers/jwt-service.js');
const nodemailer = require('nodemailer');
const restorePasswordKeyModel = require('../../../db/models/user_models/restore-password-key-model.js');
require('dotenv').config();
const sequelizeOperators = sequelize.Op;
const salt = bcrypt.genSaltSync(10);
```

```
const transport = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: process.env.EMAIL_NAME,
    pass: process.env.EMAIL_PASSWORD,
  },
});
function createAccount(registrationRequest, response) {
  const password = bcrypt.hashSync(registrationRequest.password, salt);
  userModel.user.create({
    login: registrationRequest.login,
    email: registrationRequest.email,
    password,
  });
  response.status(200).send('Вы успешно зарегестрировались');
}
function exists(login, email) {
  return userModel.user.findOne({
    raw: true,
    where: {
      [sequelizeOperators.or]: [{login}, {email}],
    },
  });
}
function updatePersonalData(request, response) {
  userModel.user.update({first_name: request.body.firstName,
    second_name: request.body.secondName,
    birthday: request.body.birthday}, {
    where: {
      login: request.body.login,
    },
  });
  response.status(200).send('Данные обновлены');
}
```

ДОДАТОК Б

core\validations\message_validation\is-author-validation.js
```
function isAuthor(userLogin, authorLogin) {
  return userLogin === authorLogin;
}
module.exports = isAuthor;
```
core\validations\user_validation\date-validator.js
```
function validateDate(date) {
  const regularExpression = /([0-9]{4}-(0[1-9]|1[012])-(0[1-9]|1[0-9]|2[0-9]|3[01]))|(0000-00-00)/;
  return date.match(regularExpression) !== null;
}
module.exports = validateDate;
```
core\validations\user_validation\email-validation.js
```
const emailValidator = require('email-validator');
function validateEmail(email) {
  return emailValidator.validate(email);
}
module.exports.validateEmail = validateEmail;
```
core\validations\user_validation\is-admin-validation.js
```
function isAdmin(userRole) {
  return userRole === 1;
}
module.exports = isAdmin;
```
core\validations\user_validation\is-restore-key-exist.js
```
const restorePasswordKeyModel = require('../../../db/models/user_models/restore-password-key-model.js');
function validateRestoreKeyDoesExist(email) {
  return restorePasswordKeyModel.findOne({
    raw: true,
    where: {
      email: email,
    },
  });
}
module.exports = validateRestoreKeyDoesExist;
```
core\validations\user_validation\login-validation.js
```
function validateLogin(login) {
  return login.match(/[^a-z0-9]/gi) === null && login.length !== 0;
}

module.exports = validateRestoreKeyDoesExist;
```
core\validations\user_validation\login-validation.js
```
function validateLogin(login) {
  return login.match(/[^a-z0-9]/gi) === null && login.length !== 0;
}
module.exports.validateLogin = validateLogin;
```
core\validations\user_validation\name-validation.js
```
function validateName(name) {
  const regularExpression = /^([A-Z][a-z]+-[A-Z][a-z]+)$|^([A-Z][a-z]+)$/;
  return name.match(regularExpression) !== null || name.length === 0;
}
module.exports = validateName;
```
core\validations\user_validation\password-validation.js
```
const PasswordValidator = require('password-validator');
const schema = new PasswordValidator();
schema
    .is().min(8)
    .is().max(100)
    .has()
    .uppercase()
    .has()
    .lowercase()
    .has()
    .digits()
    .has()
    .not()
    .spaces()
    .has()
    .not()
    .symbols();
function validatePassword(password) {
  return !!(password.match(/([^a-z0-9])/gi) === null && schema.validate(password));
}
```

```
module.exports.validatePassword = validatePassword;
core\validations\user_validation\sign-in-validation.js
const mailValidation = require('./email-validation.js');
const passValidation = require('./password-validation.js');
const loginValidation = require('./login-validation.js');
function validateSignIn(loginRequest) {
  return !!((mailValidation.validateEmail(loginRequest.login)
    || loginValidation.validateLogin(loginRequest.login))
    && passValidation.validatePassword(loginRequest.password));
}
module.exports.validateSignIn = validateSignIn;
core\validations\user_validation\sign-up-validation.js
const emailValidation = require('./email-validation.js');
const passValidation = require('./password-validation.js');
const loginValidation = require('./login-validation.js');
function validateSignUp(registrationRequest) {
  return !!(emailValidation.validateEmail(registrationRequest.email)
    && passValidation.validatePassword(registrationRequest.password)
    && loginValidation.validateLogin(registrationRequest.login));
}
module.exports.validateSignUp = validateSignUp;
core\validations\user_validation\user-personal-data-validation.js
const nameValidation = require('./name-validation.js');
const dateValidator = require('./date-validator.js');
function validatePersonalData(userPersonalData) {
  return !!(nameValidation(userPersonalData.firstName)
    && nameValidation(userPersonalData.secondName)
    && dateValidator(userPersonalData.birthday));
}
module.exports = validatePersonalData;
routers\access-control-router.js
const express = require('express');
const router = express.Router();

router.use(function(req, res, next) {
  res.header('Access-Control-Allow-Origin', '*'); // update to match the domain you will make the request from
  res.header('Access-Control-Allow-Headers', '*');
  next();
});
module.exports = router;
routers\controllers\message_controllers\message-service.js
const messageModel = require('../../../db/models/message_models/message-model.js');
function createMessage(message) {
  messageModel.create({
    author_name: message.author_name,
    topic_id: message.topicId,
    date: message.creationDate,
    text: message.text,
  });
}
function showOldMessages(topicId) {
  return messageModel.findAll({
    raw: true,
    where: {
      topic_id: topicId,
    },
  });
}
function deleteMessage(messageToDeleteId) {
  messageModel.destroy({
    raw: true,
    where: {
      id: messageToDeleteId,
    },
  });
}
```

```
function updateMessage(messageToUpdateId, messageText) {
  messageModel.update({text: messageText}, {
    raw: true,
    where: {
      id: messageToUpdateId,
    },
  });
}
function findMessage(messageId) {
  return messageModel.findOne({
    raw: true,
    where: {
      id: messageId,
    },
  });
}
module.exports.createMessage = createMessage;
module.exports.showOldMessages = showOldMessages;
module.exports.deleteMessage = deleteMessage;
module.exports.updateMessage = updateMessage;
module.exports.findMessage = findMessage;
routers\controllers\message_controllers\topic-like-service.js
const topicLikeModel = require('../../../db/models/message_models/topic-likes-model.js');
const topicModel = require('../../../db/models/message_models/topic-model.js');
const topicService = require('./topic-service.js');
const sequelize = require('sequelize');
const sequelizeOperators = sequelize.Op;
const moment = require('moment');
function isFirstDayOfTheMonth() {
  const date = new Date();
  return (date.getDate() === 1);
}
function doesLikesExist(Likes) {
  return !!Likes[0];
}

function findLikes(userLogin) {
  return topicLikeModel.findAll({
    where: {
      user_login: userLogin,
    },
  });
}
function findMonthlyOrWeeklyTopicLikes(searchingType, searchingValue, topics, i) {
  return topicLikeModel.findAll({
    raw: true,
    where: {
      topic_id: topics[i].id,
      date: {
        [sequelizeOperators.gte]: moment().subtract(searchingValue, searchingType).toDate(),
      },
    },
  });
}
function updateTopTopicLikes(Likes, topicId, type) {
  if (doesLikesExist(Likes)) {
    topicModel.update({[type]: Likes.length}, {
      where: {
        id: topicId,
      },
    });
  } else {
    topicModel.update({[type]: 0}, {
      where: {
        id: topicId,
      },
    });
  }
}
```

```
function countWeeklyLikes() {
  const type = 'weekly_likes_counter';
  topicService.findAllTopics().then((topics) => {
    for (let i=0; i<topics.length; i++) {
      const topicId = topics[i].id;
      findMonthlyOrWeeklyTopicLikes('days', 7, topics, i).then((Likes)=>{
        updateTopTopicLikes(Likes, topicId, type);
      });
    }
  });
}
function countMonthlyLikes() {
  if (isFirstDayOfTheMonth()) {
    const type = 'monthly_likes_counter';
    topicService.findAllTopics().then((topics) => {
      for (let i=0; i<topics.length; i++) {
        const topicId = topics[i].id;
        findMonthlyOrWeeklyTopicLikes('months', 1, topics, i).then((Likes)=>{
          updateTopTopicLikes(Likes, topicId, type);
        });
      }
    });
  } else {
    return;
  }
}
function likeTopic(topicId, userLogin) {
  topicLikeModel.findOne({
    raw: true,
    where: {
      topic_id: topicId,
      user_login: userLogin,
    },
  }).then((topicLike) => {
    if (!topicLike) {

      const date = new Date();
      topicLikeModel.create({
        topic_id: topicId,
        user_login: userLogin,
        date: date,
      });
      topicModel.increment('likes', {
        by: +1,
        where: {
          id: topicId,
        },
      });
    } else {
      topicLikeModel.destroy({
        where: {
          topic_id: topicId,
          user_login: userLogin,
        },
      });
      topicModel.increment('likes', {
        by: -1,
        where: {
          id: topicId,
        },
      });
    }
  });
}
function countLikes(topicId) {
  return topicModel.findOne({
    raw: true,
    where: {
      topic_id: topicId,
    },
  });
}
```

```
function deleteLikes(topicId) {
  topicLikeModel.destroy({
    raw: true,
    where: {
      topic_id: topicId,
    },
  });
}
function setWeeklyInterval() {
  setInterval(
      () => countWeeklyLikes(),
      604800000,
  );
}
function setMonthlyInterval() {
  setInterval(
      () => countMonthlyLikes(),
      86400000,
  );
}
module.exports.setMonthlyInterval = setMonthlyInterval;
module.exports.setWeeklyInterval = setWeeklyInterval;
module.exports.countLikes = countLikes;
module.exports.findLikes = findLikes;
module.exports.deleteLikes = deleteLikes;
module.exports.likeTopic = likeTopic;
routers\controllers\message_controllers\topic-service.js
const topicModel = require('../../../db/models/message_models/topic-model.js');
const messageModel = require('../../../db/models/message_models/message-model');
function createTopic(topic, response) {
  topicModel.create({
    topic_name: topic.name,
    creator_name: topic.creator,
  });
  response.status(200);
}

function findAllTopics() {
  return topicModel.findAll({
    raw: true,
  });
}
function findTopic(topicId) {
  return topicModel.findOne({
    raw: true,
    where: {
      id: topicId,
    },
  });
}
function deleteTopic(topicToDeleteId) {
  topicModel.destroy({
    raw: true,
    where: {
      id: topicToDeleteId,
    },
  });
}
function clearingTopic(topicId) {
  messageModel.destroy({
    raw: true,
    where: {
      topic_id: topicId,
    },
  });
}
```

```
function findUserTopics(userLogin) {
  return topicModel.findAll({
    raw: true,
    where: {
      creator_name: userLogin,
    },
  });
}
function findTopicsOnPage(page) {
  const limit = 10;
  const offset = (page-1) * limit;
  return topicModel.findAll({
    offset,
    limit,
    raw: true,
  });
}
function findTopTopics(type) {
  const limit = 10;
  return topicModel.findAll({
    limit,
    order: [[
      type, 'DESC'],
    ],
  });
}
module.exports.findTopTopics = findTopTopics;
module.exports.findTopicsOnPage = findTopicsOnPage;
module.exports.findUserTopics = findUserTopics;
module.exports.createTopic = createTopic;
module.exports.findAllTopics = findAllTopics;
module.exports.findTopic = findTopic;
module.exports.deleteTopic = deleteTopic;
module.exports.clearingTopic = clearingTopic;
routers\controllers\user_controllers\authentification-service.js
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const userService = require('./user-service.js');
const topicLikeService = require('../message_controllers/topic-like-service.js');

require('dotenv').config();
function signUp(request, response) {
  userService.exists(request.login, request.email)
      .then((User) => {
        if (!User) {
          userService.createAccount(request, response);
          return;
        }
        response.status(409).send('Такой пользователь уже существует');
      }).catch((err) => response.status(500).send(err));
}
function signIn(request, response) {
  userService.exists(request.login, request.login)
      .then((User) => {
        if (!User) {
          response.status(400).send('Данные введены неправильно');
        } else {
          bcrypt.compare(request.password, User.password, (err, res) => {
            if (res) {
              const responseStr = `${User.login} ${User.role_id.toString()}`;
              const token = jwt.sign(responseStr, process.env.JWT_KEY);
              topicLikeService.findLikes(User.login).then((likes) =>{
                response.status(200).json({token: token, likes: likes});
              });
            } else {
              response.status(400).send('Данные введены неправильно');
            }
          });
        }
      }).catch((err) => response.status(500).send(err));
```

```
module.exports.signUp = signUp;
module.exports.signIn = signIn;
routers\controllers\user_controllers\authorization-service.js
const jwt = require('jsonwebtoken');
function checkAuthorization(autHeader) {
  if (!autHeader) {
    return false;
  }
  try {
    jwt.verify(autHeader, process.env.JWT_KEY);
  } catch (e) {
    if (e instanceof jwt.JsonWebTokenError) {
      return false;
    }
  }
  return true;
}
module.exports = checkAuthorization;
routers\controllers\user_controllers\jwt-service.js
const jwt = require('jsonwebtoken');
function getLogin(autHeader) {
  const token = jwt.decode(autHeader);
  const login = token.replace(/\s[0-1]$/, '');
  return login;
}
function getRole(autHeader) {
  const token = jwt.decode(autHeader);
  const role = token.replace(/\S+\s/, '');
  return role;
}
module.exports.getLogin = getLogin;
module.exports.getRole = getRole;
routers\controllers\user_controllers\user-service.js

const bcrypt = require('bcrypt');
const sequelize = require('sequelize');
const userModel = require('../../../db/models/user_models/user-model.js');
const passwordValidator = require('../../../core/validations/user_validation/password-validation.js');
const emailValidation = require('../../../core/validations/user_validation/email-validation.js');
const jwtService = require('../user_controllers/jwt-service.js');
const nodemailer = require('nodemailer');
const restorePasswordKeyModel = require('../../../db/models/user_models/restore-password-key-model.js');
require('dotenv').config();
const sequelizeOperators = sequelize.Op;
const salt = bcrypt.genSaltSync(10);
const transport = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: process.env.EMAIL_NAME,
    pass: process.env.EMAIL_PASSWORD,
  },
});
function createAccount(registrationRequest, response) {
  const password = bcrypt.hashSync(registrationRequest.password, salt);
  userModel.user.create({
    login: registrationRequest.login,
    email: registrationRequest.email,
    password,
  });
  response.status(200).send('Вы успешно зарегестрировались');
}
function exists(login, email) {
  return userModel.user.findOne({
    raw: true,
    where: {
      [sequelizeOperators.or]: [{login}, {email}],
    },
  });
}
```

```
function updatePersonalData(request, response) {
  userModel.user.update({first_name: request.body.firstName,
    second_name: request.body.secondName,
    birthday: request.body.birthday}, {
    where: {
      login: request.body.login,
    },
  });
  response.status(200).send('Данные обновлены');
}
function changePassword(request, response) {
  const autHeader = request.get('Token');
  const login = jwtService.getLogin(autHeader);
  const {password} = request.body;
  if (passwordValidator.validatePassword(password)) {
    const passwordToWrite = bcrypt.hashSync(password, salt);
    userModel.user.update({password: passwordToWrite}, {
      where: {
        login,
      },
    });
    response.status(200).send('Данные обновлены');
  } else {
    response.status(400).send('Неправильные данные');
  }
}
function changeEmail(request, response) {
  const autHeader = request.get('Token');
  const login = jwtService.getLogin(autHeader);
  const {email} = request.body;
  if (emailValidation(email)) {
    userModel.user.update({email}, {
      where: {
        login,
      },
    });
```

```
function findUser(userLogin) {
  return userModel.user.findOne({
    raw: true,
    where: {
      login: userLogin,
    },
  });
}
function restorePassword(request, response) {
  if (passwordValidator(request.body.password)) {
    userModel.user.update({password: request.body.password}, {
      raw: true,
      where: {
        email: request.body.email,
      },
    });
    deleteRestoreKey(request.body.email);
    response.status(200).send('Пароль обновлен');
  } else {
    response.status(400).send('Неправильно введнный пароль');
  }
}
function sendEmail(addressee, subject, text) {
  const message = {
    from: process.env.EMAIL_NAME,
    to: addressee,
    subject: subject,
    text: text,
  };
  transport.sendMail(message, function(err, info) {
    if (err) {
      console.log(err);
    } else {
      console.log(info);
    }
  });
}

function createRestoreKey(email) {
  const key = bcrypt.hashSync(email, salt);
  restorePasswordKeyModel.create({
    key: key,
    email: email,
  });
  return key;
}
function deleteRestoreKey(email) {
  restorePasswordKeyModel.destroy({
    where: {
      email: email,
    },
  });
}
module.exports.deleteRestoreKey = deleteRestoreKey;
module.exports.createRestoreKey = createRestoreKey;
module.exports.sendEmail = sendEmail;
module.exports.restorePassword = restorePassword;
module.exports.createAccount = createAccount;
module.exports.exists = exists;
module.exports.updatePersonalData = updatePersonalData;
module.exports.changePassword = changePassword;
module.exports.changeEmail = changeEmail;
module.exports.findUser = findUser;
routers\message_routers\create-topic.js
const express = require('express');
const router = express.Router();
const topicService = require('../controllers/message_controllers/topic-service.js');
const jwtService = require('../controllers/user_controllers/jwt-service.js');
const Topic = require('../../db/db_objects/message_db_objects/topic.js');
router.post('/', (request, response) => {
  const autHeader = request.get('Token');
  const login = jwtService.getLogin(autHeader);
  const topic = new Topic(request.body.topicName, login);
  topicService.createTopic(topic, response);
});
```