Державний вищий навчальний заклад

Ужгородський національний університет

Факультет інформаційних технологій

Кафедра програмного забезпечення систем

**ЛАБОРАТОРНА РОБОТА № 7**

**Тема:** Бек-енд частина

Виконав студент:

ІІІ курсу спеціальності «Інженерія програмного забезпечення»

Нищий Богдан Вячеславович

**Ужгород-2024**

**Some news project**

Метою даної лабораторної роботи є реалізувати API для отримання необхідних даних на фронтенд частині та для авторизації.

Для досягнення даної мети, студент може обрати один з варіантів, по бажанню, але є одне але, максимально оцінку 10 отримає той студент, який реалізує бек-енд варіантом 1 (власний node/express сервер):

1. власний node/express сервер (максимальна оцінка - 10 балів)

2. Firebase (максимальна оцінка - 8 балів)

3. Готові мок-api рішення, на кшталт [mockaroo](https://www.mockaroo.com/" \t "_blank) - (максимальна оцінка - 6 балів)

По завершенню роботи студент повинен:

1. Створити пул-реквест

2. Оформити відповідного взірця [звіт](https://moodle.uzhnu.edu.ua/pluginfile.php/85716/mod_assign/intro/report_template.docx)

**Хід роботи**

Проєкт some news включає в себе логіку авторизації та реєстрації, перегляд та створення статей. Для отримання статей використовується news.api – сервіс, що дає змогу безкоштовно отримувати набори новин за різними параметрами у режимі розробки. Для усіх функцій проєкту було створено сервер, який обробляє запити з фронт-енд частини програми. Серверна частина проєкту займається валідацією даних перед авторизацією або авторизацію, і відповідно, організовує сам процес входу чи створення користувача.

Бек-енд частина проєкту приймає запити від фронт-енда не лише при авторизації чи реєстрації. Сервер використовує маршрути для отримання новин з news.api, для отримання даних користувача, даних про статті користувача. Він приймає шлях, на який фронт-енд робить запит, і отримує дані з нього, які в результаті передаються клієнту.

Сервер some news project створено з допомогою node.js та express.js. В проєкті не використовуються готові mock-api рішення та firebase.

Express – фреймворк для створення сервера node.js.

Cors – модуль для створення запитів з різних доменів.



Рис. 1. Підключена залежність для express.

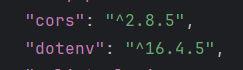


Рис. 2. Підключені залежності cors та dotenv.

Dotenv – це бібліотека, яка дозволяє створювати та використовувати змінні середовища у файлі, де зберігається інформація про дані бази даних, секретні ключі тощо. Файл .env який створений в корені проєкту зберігає всі ці дані та не додається в git репозиторій.

Сервер винесений в окремий файл:

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

**const** cors = require('cors');

**const** connectDB = require('../config/db.config');

**const** authRoutes = require('./routes/authRoute');

**const** newsApiRoutes = require('./routes/newsAPI/newsApi');

**const** userRoutes = require('./routes/userRoute/userRoute');

**const** articleRoutes = require('./routes/articleRoute/articleRoute');

connectDB();

require('dotenv').config();

**const** app = express();

app.use(cors());

app.use(express.json());

app.get('/', (request, res) => {

res.send('API is ok');

});

app.use('/api/auth', authRoutes);

app.use('/api/newsapi', newsApiRoutes);

app.use('/api/users', userRoutes);

app.use('/api/articles', articleRoutes);

**const** PORT = process.env.PORT || **5000**;

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

Тут підключаються всі необхідні залежності, бібліотеки та файли, що потрібні серверу express.

* authRoutes – відповідає за авторизацію та реєстрацію користувачів;
* newsApiRoutes – відповідає за отримання новин з news.api;
* userRoutes – відповідає за редагування даних користувача (змінна паролю, біографії, аватарки профілю);
* articleRoutes – відповідає за так звані «community/user articles», тобто користувацькі статті. Їх створення, видалення, перегляд.

Відповідно, кожному файлу задається певний маршрут сервера:

app.use('/api/auth', authRoutes);

app.use('/api/newsapi', newsApiRoutes);

app.use('/api/users', userRoutes);

app.use('/api/articles', articleRoutes);

Проте спочатку цей сервер створюється та до нього підключають допоміжні бібліотеки та можливості:

**const** app = express();

app.use(cors());

app.use(express.json());

Порт сервера, змінна якого лежить в .env файлі, використовується для фактичного запуску сервера та встановленні очікування на запити.

**const** PORT = process.env.PORT || **5000**;

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

Для того, щоб перевірити, чи дійсно сервер очікує на запити, та відповідає на них, можна перейти по відповідному шляху <http://localhost:5000/>, та перевірити відповідь сервера:

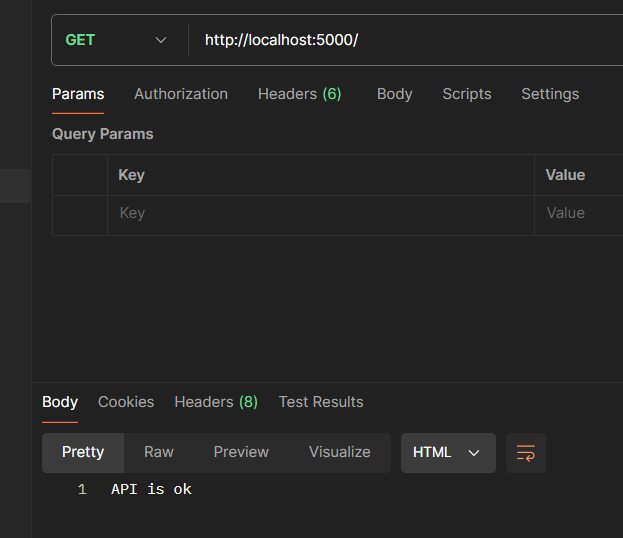


Рис. 3. Отримання відповіді сервера.

Файли маршрутів, як і сам сервер, знаходяться в окремій директорії server, проте на відміну від server.js лежать в директорії routes.

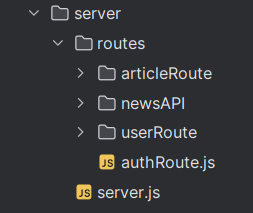


Рис. 4. Директорія server.

Для авторизації користувача використовується authRoutes за маршрутом /api/auth.

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

**const** bcrypt = require('bcryptjs');

**const** jwt = require('jsonwebtoken');

**const** User = require('../../src/models/User');

**const** regAndLog = require('../../src/validations/regAndLog');

**const** { validationResult } = require('express-validator');

**const** router = express.Router();

require('dotenv').config();

**const** JWT\_SECRET = process.env.SECRET\_KEY;

Тут у файлі підключаються express та модель даних, створена для управління та збереження даних користувачів у базі даних mongodb. Після цього, відповідно, створюється раутер. Бібліотека bcrypt використовується для хешування паролів та порівняння паролів при авторизації. Для авторизації також використовується jwt токен, який також підключається для authRoutes. regAndLog створений для валідації даних користувача перед його реєстрацією.

JWT\_SECRET береться з .env файлу. З допомогою нього шифрується jwt токен користувача.

Отже, для роботи з маршрутами authRoutes створюється раутер. Він визначає конкретні маршрути, на які робить запити клієнт.

Коли користувач вводить дані у форму авторизації та натискає кнопку «Login» фрон-енд частина, після перевірки даних, робить запит на сервер.

**const** res = await axios.post('http://localhost:5000/api/auth/login', { email, password });

Сюди передаються дані, які надав користувач – електрона пошта та пароль до акаунта.

router.post('/login', async (request, res) => {

**const** { email, password } = request.body;

**try** {

**const** user = await User.findOne({ email });

**if** (!user) {

**return** res.status(**400**).json({ msg: 'Invalid credentials' });

}

console.log(user);

bcrypt.compare(password, user.password, **function** (error, result) {

**if** (error) {

console.log('Server error');

**return** res.status(**500**).send('Server error');

}

**if** (result) {

console.log('Successfully logged in');

**const** token = jwt.sign({ id: user.\_id.toString() }, JWT\_SECRET, { expiresIn: '1h' });

**return** res.json({ token });

} **else** {

console.log('Invalid credentials');

**return** res.status(**400**).json({ success: **false**, message: 'Invalid credentials' });

}

});

} **catch** (error) {

console.error(error.message);

res.status(**500**).send('Server error');

}

});

В самій функції, яка приймає запит на авторизацію, електрона пошта та пароль беруться з request – це є дані, які надав клієнт. В конструкції try catch в базі даних в колекції user шукається документ з електронною поштою, яку вказав користувач. Якщо такий є, сервер переходить до перевірки паролю, з допомогою bcrypt.compare() із захешованим в базі даних паролем користувача.

Результат перевірки передається в res та повертається клієнту. Якщо паролі співпадають, створюється jwt токен, який і повертається фронт-енду. Помилки обробляються та повертаються клієнту також.

Якщо брати до уваги маршрут /api/articles/, який використовує articleRoutes, то на початку файлу створюється відповідний раутер, для визначення маршрутів articleRoutes.

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

**const** Article = require('../../../src/models/Article');

**const** User = require('../../../src/models/User');

**const** router = express.Router();

Тут у файлі підключаються express та моделі даних, створенні для управління та збереження даних користувачів та статей у базі даних mongodb. Після цього, відповідно, створюється раутер.

Маршрут /api/articles/ реалізує декілька підмаршрутів, на які клієнт робить запит з метою отримання чи редагування деяких даних статей.

Перший з них: ‘/save-article’ реалізує функціонал створення користувацьких статей.

Уявімо ситуацію, за якої користувач захотів створити свою статтю. Для цього, в браузері він переходить на маршрут /new-article.

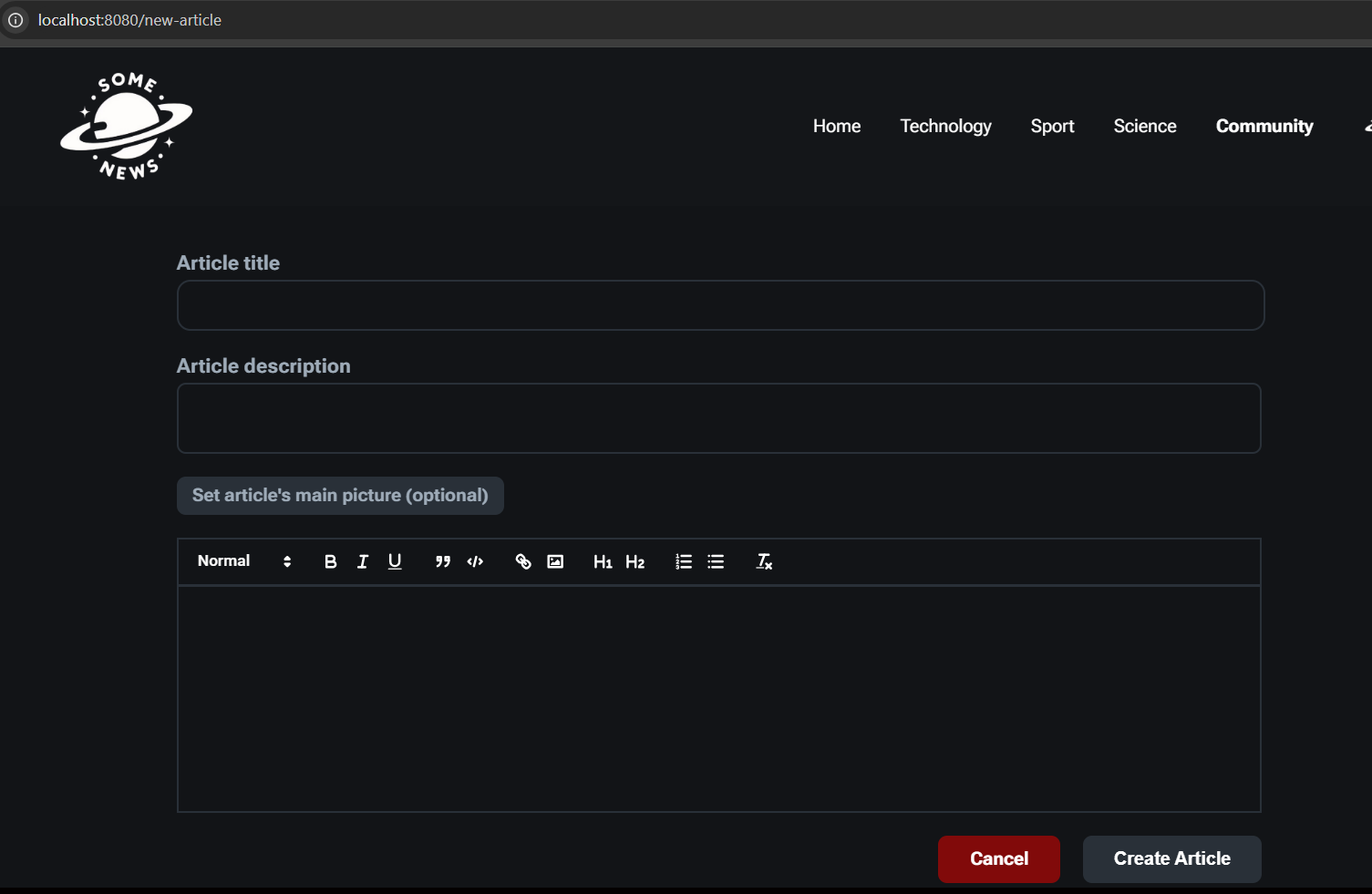


Рис. 5. Вигляд сторінки створення статті.

Для створення статті необхідно вказати її назву, описати та, відповідно, створити контент у текстовому редакторі. Для створення контенту було обрано quill rich text editor.

За бажанням користувач може додати головне фото статті, яке буде висвітлюватись в сторінці community та на сторінці userFullArticle.

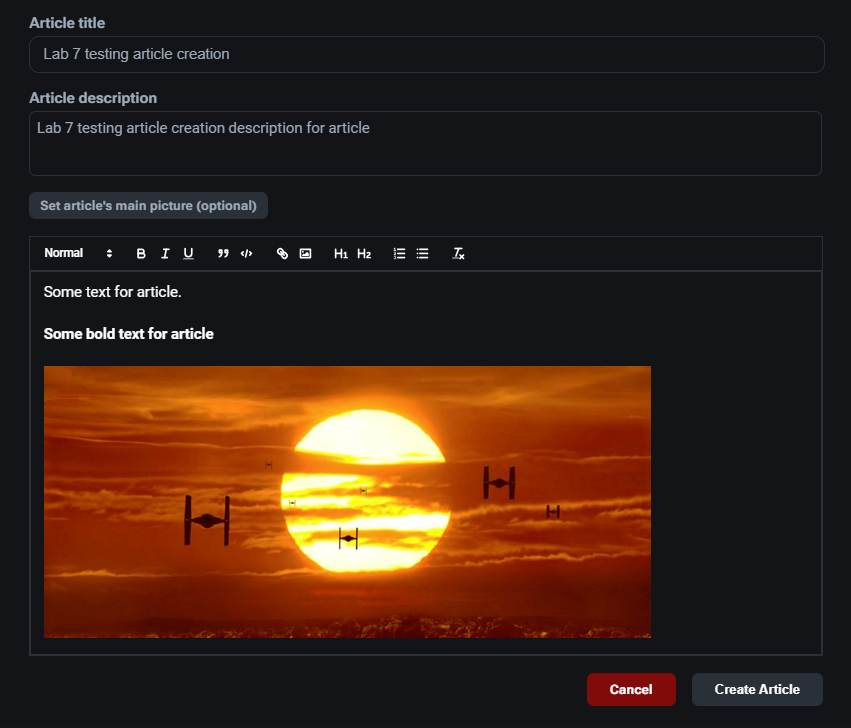


Рис. 6. Створення статті.

Після натискання кнопки «Create article» фронт-енд частина проводить необхідну перевірку даних та врешті робить запит на сервер, точніше на маршрут /save-article.

**const** response = await axios.post('http://localhost:5000/api/articles/save-article', {

title: articleTitle,

description: articleDescription,

mainPicture: base64String,

content: articleContent,

author: user.id,

});

Сервер приймає цей запит та передає його відповідному маршруту.

router.post('/save-article', async (req, res) => {

**const** { title, description, mainPicture, content, author } = req.body;

**try** {

**const** articleDoc = await Article.create({

title,

description,

mainPicture,

content,

author,

repostsCount: **0**,

});

**const** article = await articleDoc.save();

console.log(article);

console.log('All done!');

**return** res.status(**200**).json({ success: **true**, message: 'Article saved successfully' });

} **catch** {

res.status(**400**).json({ success: **false**, message: 'Failed to create article' });

}

});

Тут раутер файлу articleRoutes приймає новий запит /save-article, та приймає дані з запиту (req). Змінна res використовується для повернення відповіді.

Дані, які передав клієнт, отримуються із об’єкта запиту. В конструкції try catch створюється документ моделі Articles. Туди вносяться всі дані, надані клієнтом, а також ті, які встановлюються за замовчуванням.

З допомогою метода save() об’єкт статті зберігається в базі даних, у відповідну колекцію.

Якщо збереження успішне, маршрут повертає статус 200 (успіх) та повідомлення про успішне створення статті.

Якщо ж під час збереження статті виникає помилка, маршрут повертає повідомлення про це.



Рис. 7. Створена стаття в колекції articles.

Тепер, користувач може переглянути створену статтю, перейшовши в браузері по маршруту ‘user-articles’.

Для отримання користувацьких статей також робиться запит на сервер, який бере дані про статті з бази даних та повертає їх клієнту.

**const** response = await axios.get('http://localhost:5000/api/articles/get-articles', {

params: { userId: user.id },

});

Тут клієнт робить запит на отримання статей авторизованого користувача. Цей запит надсилається в /get-articles, який, очевидно, є запитом GET.

router.get('/get-articles', async (req, res) => {

**const** userId = req.query.userId;

**try** {

**const** articles = await Article.find({ author: userId }).sort({ createdAt: -**1** });

**if** (articles.length === **0**) {

**return** res.status(**200**).json({

message: 'No articles found',

articles: [],

});

}

**return** res.status(**200**).json(articles);

} **catch** (error) {

console.log(error);

**return** res.status(**400**).json({

message: 'Failed to get articles from the database',

});

}

});

Раутер створює маршрут, який приймає req та поверне res. З запиту він бере дані про id користувача. В конструкції try catch, з допомогою метода find, отримуються всі статті в яких id автора співпадає з id користувача. Знайдені статті сортуються у спадному порядку, щоб повернути клієнту найсвіжіші. Якщо довжина масиву, який надала база даних, дорівнює нулю, отже користувач ще не має статей.

З допомогою res сервер повертає відповідні дані або повідомлення. Якщо виникла помилка, вона також повертається клієнту.

Фронт-енд вже займається виводом користувацьких статей користувачу.

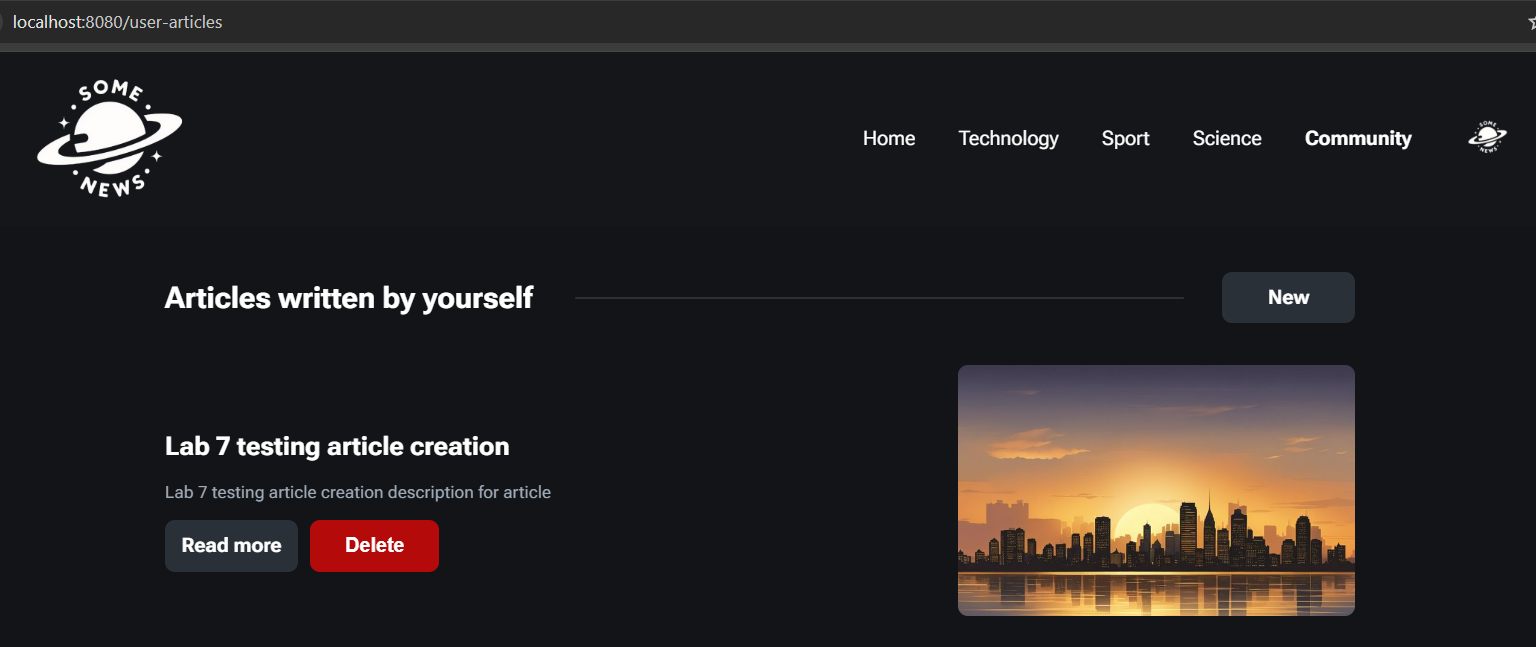


Рис. 8. Стаття, створена користувачем.

Створімо ще одну статтю і погляньмо, в якому порядку вони виводяться користувачу.

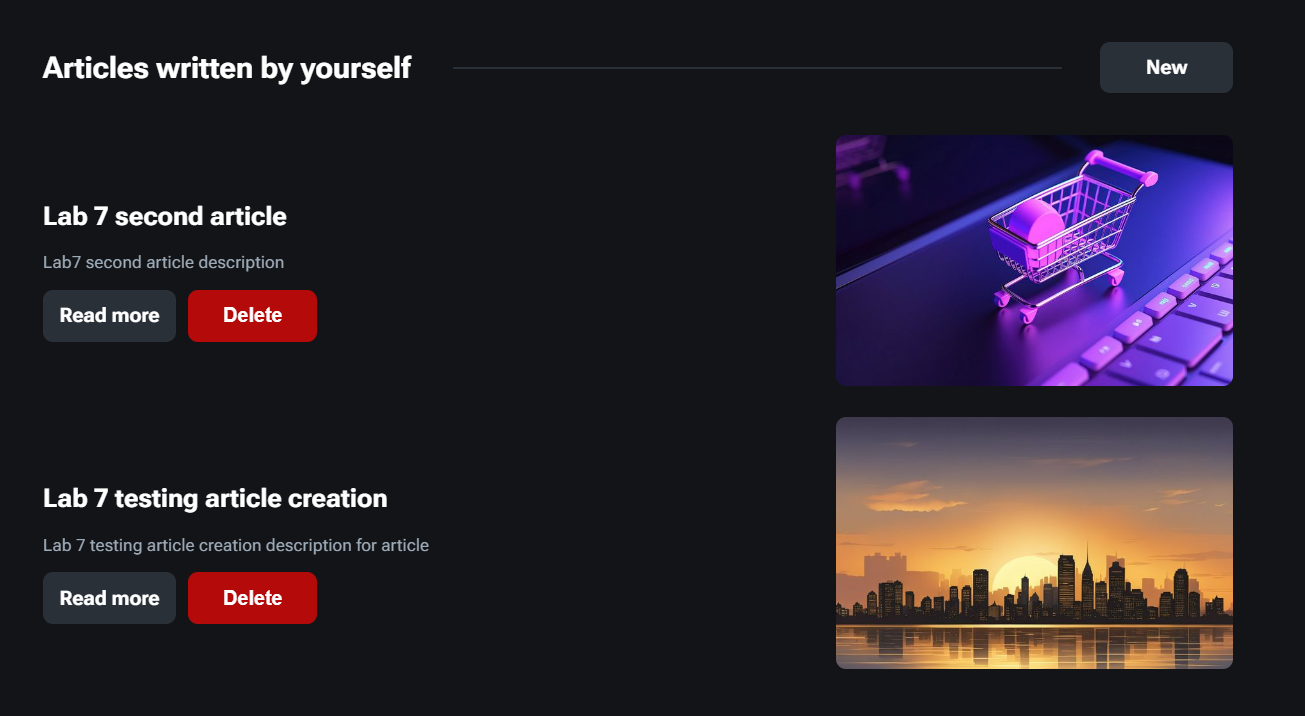


Рис. 9. Виведенні статті користувача.

Як бачимо, найвище знаходяться статті, що були опубліковані пізніше.

Всі інші маршрути та їх можливості опубліковані в [репозиторії проєкту some news в github](https://github.com/nyshchyibohdan/SomeNewsProject).

Висновок: Було створено власний node.js сервер, з використанням express та інших допоміжних бібліотек. Було підключено базу даних, створено додаток сервера та підключені відповідні маршрути для авторизації, news.api, управління даними користувача та його статей. Виконано декілька запитів для авторизації користувача, створення та виведення його статей.