Державний вищий навчальний заклад
Ужгородський національний університет
Факультет інформаційних технологій
Кафедра програмного забезпечення систем

ЛАБОРАТОРНА РОБОТА №7

Тема: Бекенд частина.

Виконав:
студент 3 курсу
спеціальності 121 «Інженерія
програмного забезпечення 3»
Лях Іван Іванович

Ужгород 2024

Мета роботи: розробити бекенд частину проекту.

Завдання до роботи: Розробити власні або використати вже наявні рішення для бекенд.

Хід роботи:

1. Бекенд проекту:

Для створення бекенд частини проекту було написано власні рішення з використанням node, express та бази даних mongoDB. Отже, бекенд частина проекту включає в себе рішення для верифікації користувача, моделі події та користувача, власноруч написані функції для валідації даних у запиті з використанням бібліотеки express-validator.

Основний серверний файл застосунку, в якому відбувається з'єднання з базою даних та визначаються роути запитів:

Код компоненту:

```
import path from 'node:path';
import { fileURLToPath } from 'node:url';
import cors from 'cors';
import express from 'express';
import connectDB from '../Config/db.config.js';
import authRoutes from './routes/authRoutes.js';
import eventRoutes from './routes/eventRoutes.js';
const app = express();
const PORT = process.env.PORT || 5000;
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
await connectDB();
app.use(express.json());
app.use(cors());
app.use('/api/', authRoutes);
app.use('/api/event', eventRoutes);
```

```
app.use(express.static(path.join(__dirname, 'dist')));
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, 'dist', 'index.html'));
});
app.listen(PORT, () => {
  console.log(`Сервер запущено на порту ${PORT}`);
});
```

Після успішного запуску серверу в терміналі відображається повідомлення про те, що сервер запущено і вказується порт, на якому він працює.

Налаштування бази даних:

Код компоненту:

```
import dotenv from 'dotenv';
import mongoose from 'mongoose';
dotenv.config();
const connectDB = async () => {
  try {
    await mongoose.connect(process.env.DB_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
      tls: true,
      tlsAllowInvalidCertificates: true,
    });
    console.log('DB is working');
  } catch (error) {
    console.error('DB error:', error);
    process.exit();
  }
};
export default connectDB;
```

В налаштуваннях бази даних було створено функцію для її запуску. В саму функцію await.mongoose.connect ми передаємо посилання на кластер бази даних, яке знаходиться в файлі .env. Після успішного встановлення з’єднання з базою даних виводимо відповідне повідомлення про те, що база даних працює, і навпаки, якщо не працює, виводимо повідомлення з помилкою.

Як зазначалося раніше, в головному файлі серверу, ми визначали роути для запитів. Одним із таких роутів є роут з назвою authRoutes. Тут знаходяться всі запити до бази даних, пов’язані з користувачем. Тобто це є запити на реєстрацію користувача, авторизацію, отримання даних про користувача та їх оновлення. Також відповідно було імпортовано модель користувача, яка містить властивості, притаманні запису про нього.

Для валідації введених даних на реєстрацію було створено окремий файл userValidation, який робив перевірку на правильність введених даних під час запиту.

Код компоненту:

```
import process from 'node:process';
import bcrypt from 'bcrypt';
import express from 'express';
import { validationResult } from 'express-validator';
import jwt from 'jsonwebtoken';
import multer from 'multer';
import { User } from '../Models/user.js';
import { registerValidation } from '../validation/userValidation.js';
const router = express.Router();
const upload = multer();
router.post('/register', registerValidation, async (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
```

```
            return res.status(400).json({ errors: errors.array() });
        }
        const { fullName, email, password, dateOfBirth } = req.body;
        try {
            const exists = await User.findOne({ email });
            if (exists) {
                return res.status(400).json({ error: 'Користувач вже існує.' });
            }
            const newUser = new User({
                fullName,
                email,
                password,
                dateOfBirth,
            });
            await newUser.save();
            res.status(201).json({ message: 'Користувач успішно створений.' });
        } catch {
            res.status(500).json({ message: 'Помилка сервера.', error:
error.message });
        }
    });
    router.post('/login', async (req, res) => {
        const { email, password } = req.body;
        try {
            const user = await User.findOne({ email });
            if (!user) {
                return res.status(400).json({ message: 'Неправильний email або
пароль.' });
            }
```

```
            const isMatch = await bcrypt.compare(password, user.get('password'));
            if (!isMatch) {
                return res.status(400).json({ message: 'Неправильний email або пароль.' });
            }
            const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, { expiresIn: '3h' });
            res.status(200).json({ message: 'Авторизація успішна.', token });
        } catch (error) {
            res.status(500).json({ message: 'Помилка сервера.', error });
        }
    });
    router.get('/users/:id', async (req, res) => {
        try {
            const userId = req.params.id;
            console.log(userId);
            const user = await User.findById(userId).select(-'password');
            if (!user) {
                return res.status(404).json({ message: 'Користувача не знайдено' });
            }
            res.status(200).json(user);
        } catch (error) {
            res.status(500).json({ message: 'Помилка сервера', error: error.message });
        }
    });
    router.put('/users/:id', upload.single('avatar'), async (req, res) => {
        try {
```

```
            const { fullName, bio, tags } = req.body;
            const tagsArray = typeof tags === 'string' ? tags.split(',').map((tag) =>
tag.trim()) : tags;
            const updateData = {
               fullName,
               bio,
               tags: tagsArray,
            };
            if (req.file) {
               updateData.avatar = req.file.buffer.toString('base64');
            }
            console.log('Updating user with ID:', req.params.id);
            console.log('Update data:', updateData);
            const user = await User.findByIdAndUpdate(req.params.id,
updateData, {
               new: true,
               runValidators: true,
            });
            if (!user) {
               return res.status(404).json({ message: 'Користувача не знайдено'
});
            }
            console.log('Updated user:', user);
            res.json(user);
         } catch (error) {
            console.error('Error during ProfilePage update:', error);
            res.status(500).json({ message: 'Помилка при оновленні даних',
error: error.message });
         }});
```

export default router;

Таким самим чином було створено роут для подій, який включав створення подій, пошук, додавання та видалення користувача з і до списку учасників події відповідно. Також кожна властивість має властивість самовидалитися, якщо кінцева дата події пройшла. Для самовидалення події є окрема функція checkExpiredEvents, яка раз в одну годину перевіряла кінцеву дату події та теперішню, і якщо час минув, подія видаляється, а для всіх учасників встановлюється значення відвіданих подій + 1.

Код компоненту:

```
import express from 'express';
import mongoose from 'mongoose';
import multer from 'multer';

import { verifyToken } from '../middleware/authMiddleware.js';
import { Event } from '../Models/event.js';
import { createEventValidation } from '../validation/eventValidation.js';

const router = express.Router();
const upload = multer();
router.get('/search', verifyToken, async (req, res) => {
  try {
    const searchQuery = req.query.q || '';
    const regex = new RegExp(searchQuery, 'i');
    const events = await Event.find({
      $or: [{ title: regex }, { tags: { $regex: regex } }],
    }).populate('author', 'fullName');
    res.status(200).json(events);
  } catch (error) {
```

```
            res.status(500).json({ message: 'Помилка сервера', error:
error.message });
        }
    });

    router.post('/create', verifyToken, upload.single('image'),
createEventValidation, async (req, res) => {
        try {
            console.log('Тіло запиту:', req.body);
            const { title, description, startLocation, endLocation, startDate,
endDate, maxParticipants, tags } = req.body;
            const author = req.user.id;
            const newEvent = new Event({
                title,
                description,
                startLocation,
                endLocation,
                startDate,
                endDate,
                maxParticipants,
                tags,
                author,
            });
            if (req.file) {
                newEvent.image = req.file.buffer.toString('base64');
            }
            await newEvent.save();
            res.status(201).json({ message: 'Подію створено успішно', event:
newEvent });
```

```
    } catch (error) {
      console.error('Помилка при створенні події:', error);
      res.status(500).json({ message: 'Помилка сервера', error:
error.message });
    }
  });
  router.get('/:id', async (req, res) => {
    const { id } = req.params;
    try {
      const event = await Event.findById(id).populate('author',
'fullName').populate('participants', '_id fullName');
      if (!event) {
        return res.status(404).json({ message: 'Подію не знайдено' });
      }
      res.status(200).json(event);
    } catch (error) {
      console.error('Помилка при отриманні події:', error);
      res.status(500).json({ message: 'Помилка сервера', error:
error.message });
    }
  });
  router.put('/:id/participate', async (req, res) => {
    const { id } = req.params;
    const { userId } = req.body;
    try {
      const event = await Event.findById(id);
      if (event.participants.length >= event.maxParticipants) {
        return res.status(400).json({ message: 'Максимальна кількість
учасників досягнута' });
```

```
        }
        if (!event.participants.includes(userId)) {
          event.participants.push(userId);
          await event.save();
        }
        res.json(event);
      } catch (error) {
        console.error('Помилка при додаванні учасника:', error);
        res.status(500).send('Помилка сервера');
      }
    });
    router.put('/:id/leave', async (req, res) => {
      const { id } = req.params;
      const { userId } = req.body;

      try {
        console.log(`Запит на видалення учасника: Подія ID = ${id},
Користувач ID = ${userId}`);

        const event = await Event.findById(id);
        if (!event) {
          console.log('Подія не знайдена');
          return res.status(404).json({ message: 'Подія не знайдена' });
        }

        const userObjectId = new mongoose.Types.ObjectId(userId);
        console.log('Учасники події:', event.participants);
```

```
            const participantIndex = event.participants.findIndex((participantId) => participantId.equals(userObjectId));

            if (participantIndex === -1) {
                console.log('Користувач не є учасником події');
                return res.status(400).json({ message: 'Користувач не є учасником події' });
            } else {
                console.log(`Користувач ${userId} знайдений серед учасників, видаляємо...`);
                event.participants.splice(participantIndex, 1);
                await event.save();
                console.log('Оновлені учасники:', event.participants);
            }

            res.json(event);
        } catch (error) {
            console.error('Помилка при видаленні учасника:', error);
            res.status(500).send('Помилка сервера');
        }
    });
    const checkExpiredEvents = async () => {
      try {
            const currentDate = new Date();
            const expiredEvents = await Event.find({ endDate: { $lt: currentDate } });
            for (const event of expiredEvents) {
                await User.updateMany({ _id: { $in: event.participants } }, { $inc: { eventsAttended: 1 } });
```

```
            console.log(`Оновлено кількість відвіданих подій для учасників події з ID: ${event._id}`);
        }
        const result = await Event.deleteMany({ endDate: { $lt: currentDate } });
        console.log(`Видалено ${result.deletedCount} подій, які закінчилися.`);
    } catch (error) {
        console.error('Помилка при видаленні подій:', error);
    }
};
setInterval(checkExpiredEvents, 3_600_000);
export default router;
```

Для верифікації користувача на можливість надсилання запитів створив окрему функцію verifyToken, яка перевіряла токен на валідність:

Код компоненту:

```
import jwt from 'jsonwebtoken';
export const verifyToken = (req, res, next) => {
  const token = req.headers.authorization?.split(' ')[1];
  if (!token) {
    return res.status(401).json({ message: 'Немає доступу. Будь ласка, увійдіть у систему.' });
  }
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch {
```

return res.status(401).json({ message: 'Неправильний або прострочений токен.' });

}};

Висновок: на даній лабораторній роботі було розроблено бекенд частину проекту, за допомогою якої з’явилася можливість користувачеві оперувати з даними бази даних. Було створено роути для подій та користувачів, в яких містилися відповідні запити до бази даних, які включали в себе створення, редагування та отримання даних. Також було створено моделі для користувачів та подій, які включали поля та їх властивості для їх передачі до бази даних. Не менш важливим є валідація цих даних, яка стала можливою завдяки створенню функції, яка включала рішення бібліотеки express-validator.