Aplikacja dla wolontariuszy - AidMate

dokumentacja projektu

Autorzy:
Paweł Zaręba
Kamil Bizoń
Bogumiła Papiernik

Spis treści

1. Struktura

1.1 Endpointy

Do obsługi endpointów użyliśmy routera z ExpressJS. Kod do nich znajduje się w katalogu “routes”, w plikach “events.js”, “index.js”, “users.js”.

1.1.1 Strona główna

```
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});
```

Endpointy dotyczące wydarzeń (eventów):

1.1.2 Dostęp do wszystkich wydarzeń

```
router.get('/:eventId',
      async (req, res, next) => {

      try {
            const {eventId} = req.params;
            console.log(`Getting event with id ${eventId}`);

            const event = await EventService.getEventById(eventId);
            res.json(event);
      } catch (e) {
            res.status(500).send();
      }
      });
```

1.1.3 Dostęp do konkretnego wydarzenia po ID

```
router.get('/:eventId',
    async (req, res, next) => {

        try {
            const {eventId} = req.params;
            console.log(`Getting event with id ${eventId}`);

            const event = await EventService.getEventById(eventId);
            res.json(event);
        } catch (e) {
            res.status(500).send();
        }
    });
```

1.1.4 Tworzenie nowego wydarzenia

```
router.post('/',
    checkToken,
    async (req, res, next) => {
        console.log(`Creating new event`);

        try {
            const createdEvent = await EventService.createEvent(req.body);
            res.json(createdEvent);
        } catch (e) {
            res.status(500).send();
        }
    });
```

1.1.5 Update wydarzenia

```
router.patch('/:eventId',
    checkToken,
    async (req, res, next) => {
        const {eventId} = req.params;

        console.log(`Creating new event`);

        try {
            const updatedEvent = await EventService.updateEvent(eventId,
req.body);
            res.json(updatedEvent);
        } catch (e) {
            res.status(500).send();
        }
    });
```

1.1.6 Dodanie uczestnika do wydarzenia

```
router.post('/:eventId/join',
    checkToken,
    async (req, res, next) => {
        const {eventId} = req.params;
        const {participantUsername} = req.body;

        console.log(`Participant ${participantUsername} is joining event
${eventId}`);
```

```
        try {
            const updatedEventParticipation = await
EventService.joinEvent(eventId, participantUsername);
            res.json(updatedEventParticipation);
        } catch (e) {
            if (e instanceof AlreadyJoinedThisEvent) {
                const {status, message} = e;
                res.status(status).json({message});
            } else {
                res.status(500).send();
            }
        }
    });
```

1.1.7 Potwierdzenie uczestnictwa w wydarzeniu

```
router.post('/:eventId/confirm',
    checkToken,
    async (req, res, next) => {
        const {eventId} = req.params;
        const {participantUsername} = req.body;

        console.log(`Participant ${participantUsername} is being
confirmed at event ${eventId}`);

        try {
            const updatedEventParticipation = await
EventService.confirmEventParticipation(eventId, participantUsername);
            res.json(updatedEventParticipation);
        } catch (e) {
            if (e instanceof AlreadyConfirmedEventParticipation || e
instanceof ResourceNotFoundError) {
                const {status, message} = e;
                res.status(status).json({message});
            } else {
                res.status(500).send();
            }
        }
    });
```

Endpointy dotyczące użytkowników (user):

1.1.8 Dostęp do użytkownika po nazwie

```
router.get('/:username',
    checkToken,
    async (req, res, next) => {
        try {
            const {username} = req.params;

            console.log(`Getting user by username ${username}`);

            const conversationResponse = await
UserService.getUserByUsername(username);
            res.json(conversationResponse);
        } catch (e) {
            console.error(e.message);
            if (e instanceof ResourceNotFoundError) {
                const {status, message} = e;
                res.status(status).json({message});
            } else {
                res.status(500).send();
            }
        }
    });
```

1.1.9 Dodanie użytkownika

```
router.post('/', async (req, res, next) => {
    try {
        console.log(`Creating new user`);

        const conversationResponse = await
UserService.createUser(req.body);
        res.json(conversationResponse);
    } catch (e) {
        console.error(e.message);
        if (e instanceof UserWithThisNameAlreadyExists) {
            const {status, message} = e;
            res.status(status).json({message});
        } else {
            res.status(500).send();
        }
    }
});
```

1.1.10 Logowanie użytkownika

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

    try {
        const {username, password} = req.body;
        console.log(`Logging in user ${username}`);

        const loginResponse = await UserService.login(username,
password);
        res.json(loginResponse);
    } catch (e) {
        if (e instanceof InvalidLoginCredentialsError) {
            const {status, message} = e;
            res.status(status).json({message});
        } else {
            res.status(500).send();
        }
    }
});
```

1.1.11 Zwrócenie listy uczestników danego wydarzenia

```
router.get("/:eventId/participants", checkToken, async (req, res, next)
=> {
    const { eventId } = req.params;
    console.log(`Getting event ${eventId} participants`);
    res.status(200);
    data = await EventService.getEventParticipants(eventId);
    res.json({ data: data });
});
```

1.1.12 Wypisanie się z wydarzenia

```
router.post("/:eventId/leave", checkToken, async (req, res, next) => {
    const { eventId } = req.params;
    const { participantUsername } = req.body;
    console.log(
        `Participant ${participantUsername} is leaving event ${eventId}`
    );
    try {
        await EventService.leaveEvent(eventId, participantUsername);
        console.log("success");
        res.status(200).send();
```

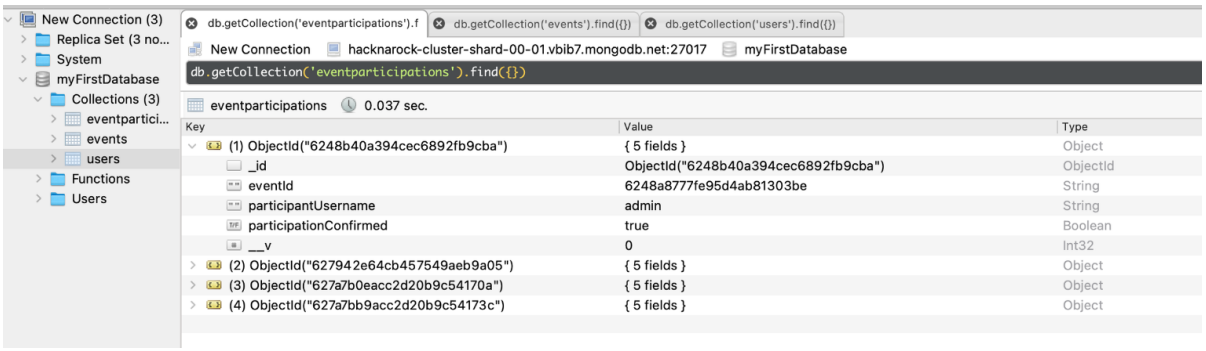
```
    } catch (e) {
        if (e instanceof IsNotParticipatingInThisEvent) {
            const { status, message } = e;
            res.status(status).json({ message });
        } else {
            res.status(500).send();
        }
    }
});
```

2. Baza danych

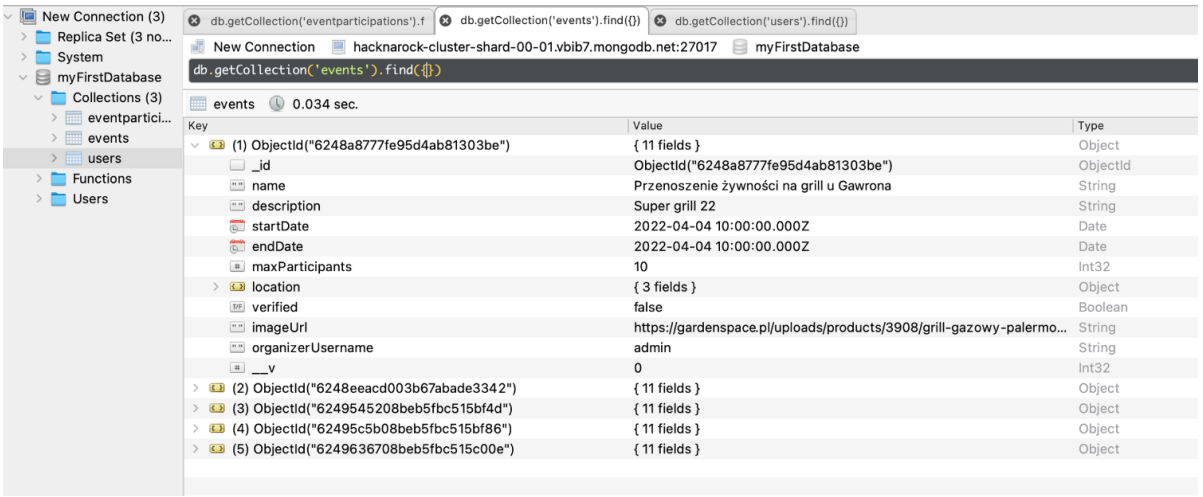
Baza danych stworzona do tego zadania zawiera trzy tabele: users (użytkownicy), events (wydarzenia) oraz eventparticipantions (uczestnictwo w wydarzeniach).

2.1 Schemat bazy

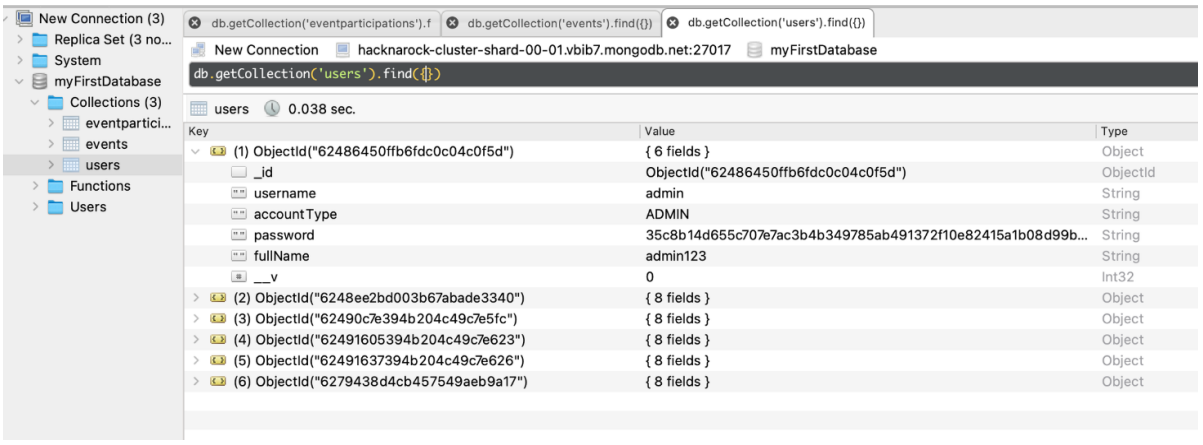
2.1.1 Tabela eventparticipantions



2.1.2 Tabela events



2.1.3 Tabela users



3. Sposób realizacji operacji w bazie danych

3.1 Modelowanie

3.1.1 EventParticipation:

```
const mongoose = require('mongoose');
const {jsonFormatterPlugin} = require("../utils/modelUtils");

const eventSchema = new mongoose.Schema({
      eventId: {
          type: String,
          required: true
      },
      participantUsername: {
          type: String,
          required: true
      },
      participationConfirmed: {
          type: Boolean,
          required: true
      }
});

eventSchema.plugin(jsonFormatterPlugin);
mongoose.model('EventParticipation', eventSchema);
```

3.1.2 Event

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

```
const {jsonFormatterPlugin} = require("../utils/modelUtils");

const eventSchema = new mongoose.Schema({
      name: {
          type: String,
          required: true,
      },
      description: String,
      startDate: {
          type: Date,
          required: true,
      },
      endDate: {
          type: Date,
          required: true,
      },
      maxParticipants: {
          type: Number,
          required: true
      },
      location: {
          city: String,
          postalCode: String,
          address: String,
      },
      verified: {
          type: Boolean,
          required: true
      },
      imageUrl: {
          type: String,
      },
      organizerUsername: {
          type: String,
          required: true
      },
});

eventSchema.plugin(jsonFormatterPlugin);
mongoose.model('Event', eventSchema);
```

3.1.3 User

```
const mongoose = require('mongoose');
const {jsonFormatterPlugin} = require("../utils/modelUtils");
```

```
const ADMIN = 'ADMIN';
const USER = 'USER';

const AccountTypes = [
      ADMIN,
      USER,
];

const userSchema = new mongoose.Schema({
      username: {
          type: String,
          required: true,
      },
      accountType: {
          type: String,
          enum: AccountTypes,
          default: USER,
          required: true,
      },
      password: {
          type: String,
          required: true,
      },
      fullName: {
          type: String,
          required: true,
      },
      email: String,
      phone: String
}
);

userSchema.plugin(jsonFormatterPlugin);
mongoose.model('User', userSchema);
```

3.2 Przykładowe operacje na bazie danych

3.2.1 Dodawanie użytkownika

```
const mongoose = require('mongoose');
require('../models/User');
const {UserWithThisNameAlreadyExists, ResourceNotFoundError} =
require("../models/errors");
const {hashPassword} = require("../utils/securityUtils");
```

```
const User = mongoose.model('User');

const createUser = async (user) => {
      const usersWithSameUsername = await User.findOne({username:
user.username});
      if (usersWithSameUsername){
          throw new UserWithThisNameAlreadyExists('User with this login
already exists')
      }

      const newUser = {
          username: user.username,
          accountType: user.accountType || 'USER',
          password: hashPassword(user.password),
          fullName: user.password || '',
          phone: user.phone || '',
          email: user.email || ''
      }

      const savedUser = await User.create(newUser);
      return savedUser;
}
```

3.2.2 Wyszukiwanie użytkownika po imieniu

```
const mongoose = require('mongoose');
require('../models/User');
const {ResourceNotFoundError} = require("../models/errors");

const User = mongoose.model('User');

const getUserByUsername = async (username) => {
      const user = await User.findOne({username: username});
      if (!user){
      throw new ResourceNotFoundError('User not found')
      }
      return user;
}
```

3.2.3 Dodawanie wydarzenia

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

```
require('../models/Event');
require('../models/EventParticipation');

const Event = mongoose.model('Event');
const EventParticipation = mongoose.model('EventParticipation');

const mapEventWithParticipants = async (event) => ({
      ...event.toJSON(),
      participants: await findParticipantsForEvents(event.id)
})

const createEvent = async (event) => {
      const eventToSave = {
          ...event,
          verified: false
      };

      const savedEvent = await Event.create(eventToSave);
      return await mapEventWithParticipants(savedEvent);
}
```

3.2.4 Wyszukiwanie uczestników wydarzenia

```
const mongoose = require('mongoose');
require('../models/Event');
require('../models/EventParticipation');

const Event = mongoose.model('Event');
const EventParticipation = mongoose.model('EventParticipation');

const findParticipantsForEvents = async (eventId) => {
      const eventParticipants = await
EventParticipation.find({eventId});

      return eventParticipants.map(({participantUsername,
participationConfirmed}) => ({
          participantUsername,
          participationConfirmed
      }))
}
```

3.2.5 Aktualizowanie wartości w bazie

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

```
require('../models/Event');
require('../models/EventParticipation');
const {
      ResourceNotFoundError,
      AlreadyConfirmedEventParticipation
} = require("../models/errors");

const EventParticipation = mongoose.model('EventParticipation');

const confirmEventParticipation = async (eventId, participantUsername)
=> {
      // Check if event exists
      const existingEventParticipation = await
EventParticipation.findOne({eventId, participantUsername});

      if (!existingEventParticipation) {
          throw ResourceNotFoundError('User has never participated in
this event')
      }
      if (existingEventParticipation.participationConfirmed) {
          throw AlreadyConfirmedEventParticipation('User has never
participated in this event')
      }

      return EventParticipation.findOneAndUpdate({
          eventId,
          participantUsername
      }, {participationConfirmed: true}, {new: true});
}
```

3.2.6 Wypisanie uczestników danego wydarzenia

```
const getEventParticipants = async (eventId) => {
    const existingEvent = await EventParticipation.find({ eventId:
eventId });
    console.log("event entries:");
    // console.log(existingEvent);
    return existingEvent.map((entry) => entry.participantUsername);
};
```