Zaawansowany Przewodnik po Node.js

Kacper Renkel

August 11, 2024

Contents

1 Node.js

Node.js to środowisko uruchomieniowe JavaScript, które umożliwia uruchamianie kodu JavaScript na serwerze. Oferuje wiele funkcji i bibliotek do budowy aplikacji serwerowych.

2 Kontrolery

W Node.js, kontrolery to zazwyczaj funkcje lub klasy, które obsługuja różne trasy w aplikacji. W Express.js, można je zdefiniować jako middleware.

2.1 Przykład Kontrolera w Express.js

```
const express = require('express');
const router = express.Router();

router.get('/users/:id', (req, res) => {
  // Kod do pobrania u ytkownika
  res.send(`User with ID: ${req.params.id}`);
});

router.post('/users', (req, res) => {
  // Kod do tworzenia u ytkownika
  res.send('User created');
});

module.exports = router;
```

Listing 1: Przykład kontrolera w Express.js

3 Serwisy

Serwisy w Node.js sa klasami lub funkcjami odpowiedzialnymi za logike biznesowa aplikacji. Zwykle sa używane do komunikacji z baza danych lub innymi zewnetrznymi źródłami danych.

3.1 Przykład Serwisu

```
class UserService {
  constructor(userModel) {
    this.userModel = userModel;
  }

  async getUser(id) {
    return this.userModel.findById(id);
  }
```

```
  async createUser(userData) {
    return this.userModel.create(userData);
  }
}

module.exports = UserService;
```

Listing 2: Przykład serwisu w Node.js

4 Repozytoria

Repozytoria w Node.js służa do zarzadzania interakcja z baza danych i czesto sa tworzone przy użyciu ORM, takich jak Sequelize lub Mongoose.

4.1 Przykład Repozytorium z Sequelize

```
const { Model, DataTypes } = require('sequelize');
const sequelize = require('./database'); // Konfiguracja bazy
    danych

class User extends Model {}

User.init({
  username: DataTypes.STRING,
  password: DataTypes.STRING,
}, {
  sequelize,
  modelName: 'User',
});

module.exports = User;
```

Listing 3: Przykład repozytorium z Sequelize

5 JWT (JSON Web Token)

JWT jest używany w Node.js do autoryzacji i uwierzytelniania. Używa sie bibliotek takich jak ‘jsonwebtoken‘ do generowania i weryfikowania tokenów.

5.1 Przykład Generowania i Weryfikowania Tokenu

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

function generateToken(payload) {
  return jwt.sign(payload, 'your-secret-key', { expiresIn: '1
    h' });
```

```
}

function verifyToken(token) {
  return jwt.verify(token, 'your-secret-key');
}

module.exports = { generateToken, verifyToken };
```

Listing 4: Przykład użycia JWT w Node.js

6 Hashowanie Haseł

Do hashowania haseł w Node.js można użyć biblioteki ‘bcrypt‘.

6.1 Przykład Hashowania Haseł

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

async function hashPassword(password) {
  const salt = await bcrypt.genSalt();
  return bcrypt.hash(password, salt);
}

async function comparePassword(password, hash) {
  return bcrypt.compare(password, hash);
}

module.exports = { hashPassword, comparePassword };
```

Listing 5: Przykład hashowania haseł przy użyciu bcrypt

7 Middleware

Middleware w Node.js to funkcje, które moga modyfikować żadania i odpowiedzi lub przeprowadzać dodatkowe operacje przed dotarciem do końcowego kontrolera.

7.1 Przykład Middleware

```
function loggerMiddleware(req, res, next) {
  console.log(‘Request made to: ${req.url}‘);
  next();
}

module.exports = loggerMiddleware;
```

Listing 6: Przykład middleware w Node.js

8 Guardy

W Node.js, implementacja zabezpieczeń jest czesto realizowana za pomoca middleware. Middleware działa podobnie do guardów w innych frameworkach.

8.1 Przykład Guardu (Middleware) w Node.js

```
function authGuard(req, res, next) {
  const token = req.headers['authorization'];
  if (token === 'valid-token') {
    next();
  } else {
    res.status(403).send('Forbidden');
  }
}

module.exports = authGuard;
```

Listing 7: Przykład guardu w Node.js

9 Interceptory

W Node.js, podobne funkcje można zrealizować poprzez middleware, które modyfikuje żadania lub odpowiedzi.

9.1 Przykład Interceptora (Middleware) w Node.js

```
function responseInterceptor(req, res, next) {
  console.log('Before handling request');
  res.on('finish', () => {
    console.log('After handling request');
  });
  next();
}

module.exports = responseInterceptor;
```

Listing 8: Przykład interceptora w Node.js

10 Protected Routes

Protected routes w Node.js sa zabezpieczane za pomoca middleware, które sprawdza autoryzacje przed dostepem do chronionych zasobów.

10.1 Przykład Chronionej Trasy

```
const express = require('express');
const app = express();
const authGuard = require('./authGuard');

app.use('/protected', authGuard, (req, res) => {
  res.send('This is a protected route');
});

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});
```

Listing 9: Przykład chronionej trasy w Node.js

11 Łaczenie sie z Baza Danych

Node.js obsługuje różne biblioteki ORM/ODM do komunikacji z baza danych. Poniżej przedstawiam konfiguracje dla dwóch popularnych bibliotek: Sequelize (dla SQL) oraz Mongoose (dla MongoDB).

11.1 Sequelize

Sequelize to popularny ORM dla Node.js, który obsługuje różne bazy danych SQL, takie jak MySQL, PostgreSQL, SQLite.

11.1.1 Przykład Konfiguracji Sequelize

```
const { Sequelize } = require('sequelize');

const sequelize = new Sequelize('database', 'username', '
   password', {
  host: 'localhost',
  dialect: 'mysql', // lub 'postgres', 'sqlite', 'mssql'
});

module.exports = sequelize;
```

Listing 10: Konfiguracja Sequelize w Node.js

11.2 Mongoose

Mongoose to popularne ODM dla MongoDB. Pozwala na łatwe połaczenie z baza danych MongoDB i zarzadzanie schematami danych.

11.2.1 Przykład Konfiguracji Mongoose

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

mongoose.connect('mongodb://localhost/nest', {
   useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.error('Failed to connect to MongoDB',
      err));

module.exports = mongoose;
```

Listing 11: Konfiguracja Mongoose w Node.js