Veebiarendus

Back-End arendus

Martti Raavel

martti.raavel@tlu.ee

Tänased teemad

- Eelmise loengu meenutamine
- Andmebaasi ühendamine NodeJS-iga
- Andmebaasi päringud Node API-s
- JOIN laused MySQL-is
- Node API automaattestimine

Millest rääkisime eelmisel korral?

Andmebaasi ühendamine NodeJS-iga

MySQL NodeJS-iga

Selleks, et NodeJS saaks MySQL andmebaasiga suhelda, kasutame `mysql2` moodulit, mis võimaldab luua ühenduse MySQL andmebaasiga, saata päringuid ja saada vastuseid.

```
npm install mysql2
```

Andmebaasi ühendamine

Andmebaasi ühendamine NodeJS-iga toimub MySQL mooduli abil, mis võimaldab luua ühenduse MySQL andmebaasiga, saata päringuid ja saada vastuseid.

```
const mysql = require('mysql2');

const pool = mysql.createPool({
  host: 'localhost',
  user: 'mysql-user',
  password: 'mysql-password',
  database: 'mydatabase'
});

const promisePool = pool.promise();

module.exports = promisePool;
```

`Pool` on MySQL ühenduste haldur, mis võimaldab luua mitu ühendust ja kasutada neid.

Andmebaasi päringud teenustes

Andmebaasi päringute tegemiseks loome teenused, mis kasutavad andmebaasi ühendust ja saadavad päringuid.

```
const db = require('../db');

const getAllUsers = async () => {
  const [rows] = await db.query('SELECT * FROM users');
  return rows;
};
```

SQL-injection

SQL-injection on turvarünnak, mis võimaldab ründajal süstida pahatahtlikku SQL-koodi veebirakendusse, et manipuleerida andmebaasi päringuid ja saada juurdepääs konfidentsiaalsetele andmetele.

Parameetrilised päringud

Parameetrilised päringud võimaldavad kasutada muutujaid ja väärtusi päringutes, mis muudavad päringud dünaamiliseks ja turvalisemaks. ,

Parameetrilised päringud aitavad vältida SQL-injection rünnakuid.

```
const getUserById = async (id) => {
  const [rows] = await db.query('SELECT * FROM users WHERE id = ?', [id]);
  return rows[0];
};

const createUser = async (username, email) => {
  const [result] = await db.query('INSERT INTO users (username, email) VALUES (?, ?)', [username, email]);
  return result.insertId;
};
```

> Pane tähele, et päringu muutuja asendatakse küsimärgiga `?` ja väärtused antakse massiivina.

Ressursside uuendamine ja kustutamine

Andmebaasi päringud võimaldavad ressursse uuendada ja kustutada.

```
const updateUser = async (id, username, email) => {
  const [result] = await db.query('UPDATE users SET username = ?, email = ? WHERE id = ?', [username, email, id]);
  return result.affectedRows;
};
```

Saab ka nii:

```
const updatedUser = async (id, user) => {
  const [result] = await db.query('UPDATE users SET ? WHERE id = ?', [user, id]);
  return result.affectedRows;
};
```

Viimase näite puhul peab `user` olema objekt, kus võtmed vastavad andmebaasi väljadele.

JOIN laused MySQL-is

MySQL JOIN-id on vahend, mis võimaldab andmeid erinevatest tabelitest pärida ja ühendada. JOIN-id võimaldavad kombineerida ridu kahest või enamast tabelist, mis on omavahel seotud primaar- ja võõrvõtmete kaudu.

JOIN-ide tüübid

- `INNER JOIN`
- `LEFT JOIN`
- `RIGHT JOIN`
- `FULL JOIN`

`INNER JOIN`

`INNER JOIN` tagastab ainult need read, millel on mõlemas tabelis vastavus.

```
SELECT column_name(s)
  FROM left_table
  INNER JOIN right_table
  ON left_table.column_name = right_table.column_name;
```

> Vasak ja parem tabel tähistavad tabelite järjekorda päringus.

`INNER JOIN` kasutamine

Tegevused koos kasutajate andmetega:

```
SELECT users.firstName, users.lastName, todos.title, todos.description, todos.is_done
  FROM todos
  INNER JOIN users
  ON todos.user_id = users.id
WHERE todos.deleted_at IS NULL;
```

`LEFT JOIN`

`LEFT JOIN` tagastab kõik read vasakust tabelist ja vastavad read paremast tabelist. Kui vastavust pole, siis täidetakse parema tabeli väljad NULL-idega.

```
SELECT column_name(s)
  FROM left_table
  INNER JOIN right_table
  ON left_table.column_name = right_table.column_name;
```

`LEFT JOIN` kasutamine

Tegevused koos kasutajate andmetega:

```
SELECT users.firstName, users.lastName, todos.title, todos.description, todos.is_done
  FROM todos
  LEFT JOIN users
  ON todos.user_id = users.id
WHERE todos.deleted_at IS NULL;
```

> Nagu näha, on `LEFT JOIN` kasutamine sarnane `INNER JOIN` -iga, vahe tekib sisse siis, kui tegemist on andmetega, millel pole vastavust.

`RIGHT JOIN`

`RIGHT JOIN` tagastab kõik read paremast tabelist ja vastavad read vasakust tabelist. Kui vastavust pole, siis täidetakse vasaku tabeli väljad NULL-idega.

`FULL JOIN`

FULL JOIN tagastab kõik read, kui on vastavus vasakus või paremas tabelis. MySQL-is ei ole otsest `FULL JOIN` toetust, kuid seda saab saavutada `UNION` abil.

Express API automaattestimine

Testimine

Testimine on protsess, mille käigus hinnatakse tarkvara kvaliteeti, kontrollides selle funktsionaalsust, jõudlust ja turvalisust. Testimise eesmärk on leida vead ja puudused ning kinnitada, et tarkvara töötab ootuspäraselt ja vastab määratletud nõuetele.

Testimise liigid

- Ühiktestimine (Unit Testing)
- Integratsioonitestimine (Integration Testing)
- Süsteemitestimine (System Testing)
- Vastuvõtutestimine (Acceptance Testing)
- jne

Testimise meetodid

- Käsitsi testimine
- Automaatne testimine

Node.js testiraamistikud

- Mocha
- Jest
- SuperTest
- jne

Mocha

Mocha on paindlik ja lihtne testiraamistik Node.js jaoks, mis võimaldab teste kirjutada ja käivitada.

Mocha paigaldamine ja kasutamine

```
npm install mocha
```

Loo projekti juurkausta kataloog nimega `test` ja selle sisse testifail, näiteks `test.js`.

Mocha testide struktuur

Mocha kasutab BDD stiilis testide kirjutamiseks funktsioone `describe`, `it` ja `before`, `after`, `beforeEach`, `afterEach`.

`describe` funktsioon võimaldab teste grupeerida ja kirjeldada.

`it` funktsioon kirjeldab üksikuid teste.

Mocha testide struktuuri näide

```
describe('Testi grupi nimetus', function() {
  it('Testi kirjeldus', function() {
    // Testi kood
  });
  it('Testi kirjeldus', function() {
    // Testi kood
  });
});
```

Mocha testide näide

```
const assert = require('assert');
const sum = require('./sum');

describe('sum', () => {
  it('should return 3 when the input is 1 and 2', () => {
    assert.strictEqual(sum(1, 2), 3);
  });

  it('should return 5 when the input is 2 and 3', () => {
    assert.strictEqual(sum(2, 3), 5);
  });

  it('should return 0 when the input is 0 and 0', () => {
    assert.strictEqual(sum(0, 0), 0);
  });
});
```

> `assert` on Node.js standardne moodul, mis võimaldab teha 'väiteid'.

Chai

Chai on populaarne asertsiooniraamatukogu JavaScripti jaoks, mida kasutatakse koos testimisraamistikega nagu Mocha ja Jest.

Chai paigaldamine

```
npm install chai
```

Chai kasutamine

```
it('should return 3 when the input is 1 and 2', () => {
  expect(sum(1, 2)).to.equal(3);
});
```

Chai asertsioonistiilid

- `expect`
- `should`
- `assert`

Chai cheatsheet: https://devhints.io/chai

Expect asertsioonistiil

```
expect(object)
  .to.equal(expected)
  .to.deep.equal(expected)
  .to.be.a('string')
  .to.include(val)
  .be.ok(val)
  .be.true
  .be.false
  .to.exist
  .to.be.null
  .to.be.undefined
  .to.be.empty
  .to.be.arguments
  .to.be.function
  .to.be.instanceOf
  .to.have.property
```

Supertest

SuperTest on Node.js raamistik, mis võimaldab testida HTTP päringuid ja vastuseid.

Supertest'i eelised võrreldes teiste raamistikega

- Lihtne integreerida
- Ei vaja serveri käivitamist
- Rikkalikud päringu võimalused
- Kergesti loetavad `assertionid`
- Toetab asünkroonseid teste

Supertest paigaldamine

```
npm install supertest
```

Supertest kasutamine

```
const chai = require('chai');
const request = require('supertest');
const app = require('../app');
const { expect } = chai;

describe('GET /ping', function() {
  it('responds with json', function(done) {
    request(app)
      .get('/')
      .set('Accept', 'application/json')
      .expect('Content-Type', /json/)
      .expect(200, done);
  });
});
```

Eeltöö

Enne, kui saame hakata oma rakendusele test kirjutama, peame oma rakendust veel natukene struktureerima. Nimelt tuleb selleks, et me saaksime API-t automaatselt testida, eraldada API loomise kood ja API käivitamise kood, kuna testimine eeldab, et API on eraldi moodulina kasutatav.

Struktureerimine

Loome oma projekti juurkausta eraldi faili `server.js`, kuhu tõstame `app.js` failist selle osa, mis käivitab serveri.

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

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

See tähendab, et `app.js` failis on ainult API loomise kood ja see tuleb sealt ka eraldi eksportida.

Supertesti testide kirjutamine

```
const request = require('supertest');
const { expect } = require('chai');
const { describe, it } = require('mocha');

const app = require('../app');

const okResponse = {
  success: true,
  message: 'pong',
};

describe('GET /ping', () => {
  it('should return 200 OK', async () => {
    const response = await request(app).get('/ping');
    expect(response.status).to.equal(200);
    expect(response.body).to.deep.equal(okResponse);
  });
});
```

Kodutöö

- Andmebaasi kasutamine oma API-s
-