

TLO-32400 Ohjelmallinen sisällönhallinta

Harjoitustyön vaihe 1

Tommi Honkanen

tommi.honkanen@tuni.fi

Valitut teknologiat ja asennus

Harjoitustyön aihe/käytettävä data ei ole vielä täysin selvä itselleni, mutta päätin valita kehitysympäristöksi Node.js ja PostgreSQL yhdistelmän, sillä en ole aikaisemmin käyttänyt Nodea mihinkään ja PostgreSQL on entuudestaan tietokantana tuttu. Toistaiseksi frontend on yksinkertaista HTML-koodia, mutta otan työhön mukaan todennäköisesti Reactin käyttöliittymää varten. Editorina ensimmäisessä vaiheessa riitti Notepad++, mutta tämä todennäköisesti vaihtuu Visual Studioon.

Noden hankinta koneelle oli Windowsia käyttäen yksinkertainen .exe-asennusohjelman ajo. Noden mukana koneelle asentui myös npm, jonka kautta tarvittavat kirjastot saa kätevästi koneelle komennolla npm install. PostgreSQL koneelta löytyi entuudestaan.

Sovelluksen rakenne ja toiminta

Sovellus on yksinkertaisuudessaan index.js-tiedosto eli Javascriptiä, jota ajetaan Noden sisällä. Node luo tästä serverin, jonka tarjontaa voi lukea localhostilla tai halutussa tuotantoympäristössä. Node-serveriin voi liittää tietokantoja kirjastoja käyttämällä, ja liittämällä kirjastot oikein index.js-tiedostoon serveri ottaa yhteyttä määriteltyyn kantaan, ja kannalle voi tehdä pyyntöjä. Omassa toteutuksessa PostgreSQL liitetään serveriin seuraavasti:

```

1
2   var express = require('express');
3
4   var path = require('path')
5   var bodyParser = require('body-parser');
6
7   var cons = require('consolidate')
8   var dust = require('dustjs-helpers')
9   var pg = require('pg')
10  var port = 8000
11
12  //DB connection string
13  var connect = "postgres://me:password@localhost/ohsiha";
14
15
16  function getClient(){
17
18      var client = new pg.Client(connect , function(err, client, done){
19          if (err) {
20              return console.error('error fethcing data',err);
21          }
22          });
23      return client;
24  }
25

```

Muuttuja pg on postgres-Nodekirjaston import, joka ilmaistaan Nodessa require-kutsulla. Tietokanta-asiakkaan voi luoda erillistä yhdistämismuuttujaa käyttämällä, jossa on tietokantakäyttäjän kirjautumistiedot, osoite sekä tietokannan nimi.

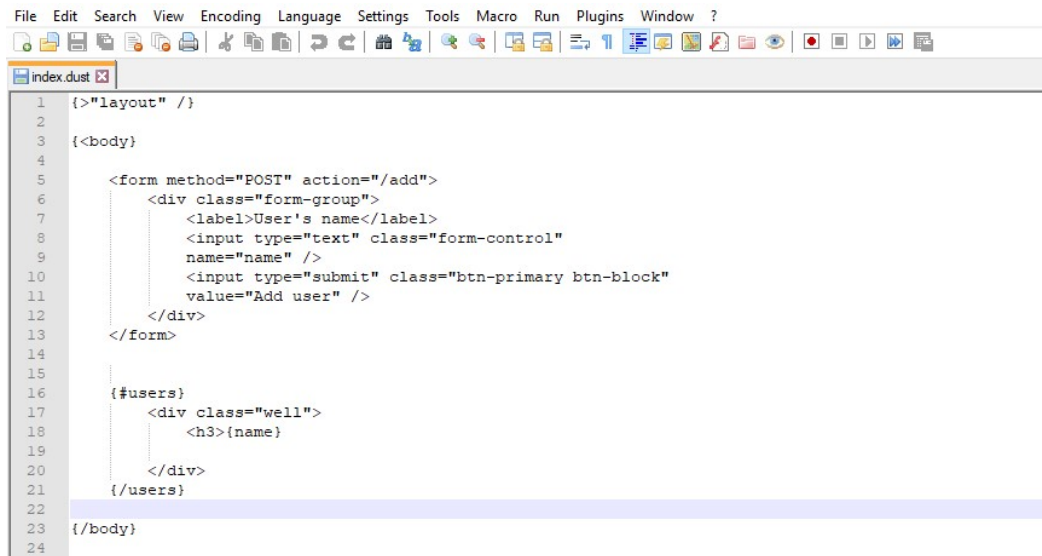
```

47
48  app.post('/add', function(req, res) {
49
50      client = getClient();
51      client.connect();
52      console.log("Trying to add new user");
53      client.query('INSERT INTO users(name) VALUES ($1)',
54          [req.body.name]);
55
56      res.redirect('/');
57      console.log("Added new user");
58  });
59
60
61  app.listen(8000, function(){
62      console.log('server started on port 8000');
63  })
64
65  app.get('/', function(req, res){
66
67      client = getClient();
68      client.connect();
69      client.query('SELECT * FROM users', function(err, result) {
70          if(err) {
71              return console.error('error running query', err);
72          }
73          res.render('index', {users: result.rows});
74          client.end();
75      });
76  });
77
78

```

Tietokantaa hyödynnetään ylläolevilla funktioilla, joissa tietokantaan liitetään asiakas, ja asiakas suorittaa pyyntöjä kohteena olevaan kantaan. Testausta varten loin kantaan taulun käyttäjistä

ja yksinkertaisen sivun, jossa HTML-formilla kantaan voi lisätä käyttäjän, ja sivu listaa kaikki kannassa olevat käyttäjät formin alle. App.post on uuden käyttäjän lisäämistä varten tehty funktio, ja app.get puolestaan tekee lukuoperaation kannan kaikille käyttäjille.



```
1  {>"layout" /}
2
3  {<body>
4
5      <form method="POST" action="/add">
6          <div class="form-group">
7              <label>User's name</label>
8              <input type="text" class="form-control"
9                  name="name" />
10             <input type="submit" class="btn-primary btn-block"
11                 value="Add user" />
12          </div>
13      </form>
14
15      {#users}
16          <div class="well">
17              <h3>{name}</h3>
18          </div>
19      {/users}
20
21  {</body>
22
23  }
```

Yllä on käytössä olevan HTML-sivun lähdekoodi. Node sisältää dust-nimisen kirjaston, jolla on mahdollista tehdä .dust-tyyppisiä HTML-tiedostoja ja renderöidä serverin lähettämää dataa muuttujiin. Tällä hetkellä olen ajanut serveriä vain localhostin kautta.

Tulevat muutokset työympäristössä ja työkaluissa

Tulevaisuudessa serveri on tarkoitus siirtää Herokuun, josta minulla on entuudestaan kokemusta. Myös käytettävän PostgreSQL-kannan voi sijoittaa Herokuun. Frontendiä varten ajattelin liittää projektiin Reactin ja koittaa tätä uutena työkaluna.

Helpot ja vaikeat asiat

- Helppoa: Node.js:n ja tarvittavien kirjastojen hankkiminen koneelle. Yhden asennusohjelman suorittaminen vakioasetuksilla ja Node-serverin käyttämän package.json-tiedoston sisältämät riippuvuudet ovat hyvin verrattavissa Djangoon ja ympäristön hallinta on yhtä helppoa. Suurin ongelma on käytännössä runsauden pula ja tietämättömyys mahtavista apukirjastoista ja työkaluista.

- Vaikeaa: PostgreSQL:n siirto Herokuun. Alun perin tarkoitukseni oli laittaa palvelu heti Herokuun pyörimään, sillä ainakin Django tapauksessa se on varsin helppoa. En kuitenkaan saanut siirrettyä PostgreSQL-kantaani Herokuun sen tarjoamilla menetelmillä johtuen hyvin oudosta tilanteesta omassa PostgreSQL-asennuksessani, jossa minulta ilmeisesti puuttuu jokin vaadittava salasana ympäristöasetuksista, mutta en myöskään saanut asetettua salasanaa PostgreSQL:n kautta.
- Helppoa: Tietokannan ja Node.js yhdistäminen. Nodeen on saatavilla hyvin paljon erilaisia tietokantaliitoksia, ja PostgreSQL:n tarvitsemat kirjastot ovat hyvin toimivia ja yksinkertaisia käyttää. Sain luettua kannasta sinne etukäteen kirjoitettua dataa melkein heti, ja jonkin ajan kuluttua myös kantaan tallentaminen onnistui.

Hyödylliset linkit

Recipe app using Node.js & PostgreSQL -videosarja Youtubessa

Devcenter.heroku.com Herokun käyttöönottoon ja erilaisten työkalujen ja palvelujen liittäminen Herokuun. Lähes kaikkeen löytyy kattavat ohjeet vaihe vaiheelta alusta loppuun.

Muut Node-aiheiset harjoitustyöpalautukset