Sukelletaan syvemmälle hakuihin.

Laitetaan book.js ja book\_model.js yhtä aikaa auki VSC:hen.

GET-pyyntö

**book\_model.js:**

getById: function(id, callback) {

    return db.query('select \* from book where id\_book=?', [id], callback);

  },

  getAll: function(callback) {

    return db.query('select \* from book', callback);

  },

getById hakee yhden kirjan pääavaimen (id\_book) perusteella:

return db.query('select \* from book where id\_book=?', [id], callback);

getAll hakee kaikki kirjat (**select \* from book**):

return db.query('select \* from book', callback);

**book.js:**

TÄÄLLÄ KUTSUTAAN edellä olleita funktioita getById ja getAll !

Jos request.params.id on tosi, eli on annettu GET:illä pääavaimen arvo, suoritetaan book.getById. SELECT-lauseen palauttama tietue (=1 kirja) tallentuu muuttujaan **dbResult**.

Jos ei anneta id:tä niin haetaan SELECT-lauseella kaikki kirjat (ilman WHERE-ehtoa lopussa):

router.get('/:id?',

 function(request, response) {

  if (request.params.id) {

    book.getById(request.params.id, function(err, dbResult) {

      if (err) {

        response.json(err);

      } else {

        response.json(dbResult);

      }

    });

  } else {

    book.getAll(function(err, dbResult) {

      if (err) {

        response.json(err);

      } else {

        response.json(dbResult);

      }

    });

  }

});

Virhetilanteessa **response.json(err)** tulostaa virheilmoituksen.

HUOM!!! Paluuarvo response.json tulostuu Postmanin alempaan ikkunaan. Esimerkiksi DELETE:n tulos:

response.json(dbResult);

{

    "fieldCount": 0,

    "affectedRows": 1,

    "insertId": 0,

    "serverStatus": 2,

    "warningCount": 0,

    "message": "",

    "protocol41": **true**,

    "changedRows": 0

}

Esim. affectedRows näyttää että yhtä riviä (tietuetta) muutettiin (tässä poistettiin).

Sen sijaan POST palauttaa request.body:n eli ne tiedot jotka syötetään:

response.json(request.body);

{

    "id\_boo": "NULL",

    "name": "Breakfast of Champions",

    "author": "Dennis Ritchie",

    "isbn": "222-222-222-222-g"

}

Huomatus parametrin välityksestä ?:llä:

return db.query('select \* from book where id\_book=?', [id], callback);

db.query suorittaa kaiken, mikä on hipsujen sisällä. Edellisen voisi kirjoittaa myös concatella

return db.query('select \* from book where id\_book='+id, [id], callback);

+id on hipsujen ulkopuolella. id:n arvoksi voisi kirjoittaa vaikka DROP TABLE book, jos olisi ilkeämielinen. Tätä sanotaan SQL-injektioksi.

Tehdään seuraavaksi toinen taulu **borrower**

**+---------------+-------------+------+-----+---------+----------------+**

**| Field | Type | Null | Key | Default | Extra |**

**+---------------+-------------+------+-----+---------+----------------+**

**| id\_borrower | int(11) | NO | PRI | NULL | auto\_increment |**

**| fname | varchar(50) | YES | | NULL | |**

**| lname | varchar(50) | YES | | NULL | |**

**| streetAddress | varchar(30) | YES | | NULL | |**

Kopioidaan Routes-hakemistossa book.js -> borrower.js ja Search – Replacella joka paikassa book -> borrower.

Kopioidaan Models-hakemistossa book\_model.js borrower\_model.js ja sama toimitus.

Huom: Tämä tietenkin toimii näin jos kenttiä on yhtä paljon. Muuten joutuu joko lisäämään/vähentämään.

Lisätään app.js

var bookRouter = require('./routes/book');

var borrowerRouter = require('./routes/borrower');

ja

app.use('/book', bookRouter);

app.use('/borrower', borrowerRouter);

Sitten Postmanilla testataan taas kaikki.

**Autentikointi**

Tehdään käyttäjä user. Routes-hakemistoon tiedosto user.js

<https://peatutor.com/express/Examples/user_control.php>

ja linkin sisältö kopioidaan sinne.

Sitten kryptaus

<https://peatutor.com/express/Examples/user_model.php>

HUOM: user\_model käyttää tietokanna user\_table-taulua:

**+----------+--------------+------+-----+---------+----------------+**

**| Field | Type | Null | Key | Default | Extra |**

**+----------+--------------+------+-----+---------+----------------+**

**| id\_user | int(11) | NO | PRI | NULL | auto\_increment |**

**| username | varchar(20) | YES | UNI | NULL | |**

**| password | varchar(255) | YES | | NULL | |**

**+----------+--------------+------+-----+---------+----------------+**

Tehdään ensin suojaus, joka suojaa vaan tämän taulun eli user.js tauluun alkuun:

const basicAuth = require('express-basic-auth');

router.use(basicAuth({users: { 'admin': '1234' }}))

Eli nyt book ja borrower-tauluihin pitäisi päästä edelleen kiinni ilman salasanaa. Mutta user\_tableen ei.