Utvikling

English Edition

Tables of Contents

Database - sqlite

Get better-sqlite3:

```
npm install better-sqlite3
```

Set up:

```
// Import:
const sqlite3 = require('better-sqlite3')

//Connect to databse, change mydb.sdb to database file name
const db = sqlite3('mydb.sdb', {verbose:console.log})


// Export:
exports.db = db
```

↑ Put the above in their approtirate places inside setup.js ↑

Workbench to Sqlite

```
//put in seperate file example: dbCreator.js
const db = require("better-sqlite3")()
const sql = `
 SQL script goes here
`
db.exec(sql);
// get sql script  from mySQL Workbench
// Tools → Catalog → Export SQlite CREATE script
```

SQL commands

```
// All
const sql = db.prepare('SELECT * FROM names')
const rows = sql.all()

// One Specific
const sql = db.prepare('SELECT * FROM names WHERE id = (?)')
const result = sql.get(1)

// Insert
const sql = db.prepare('INSERT INTO names (first,last) VALUES (?,?)');
const info = sql.run("Birger","Hansen")

// Delete
const sql = db.prepare("DELETE FROM names WHERE id=(?)");
const info = sql.run("7")

// Update
const sql = db.prepare(`
        UPDATE names
        SET name=(?)
        WHERE id=(?)`);
const info = sql.run("Spiderman",8)

// Inner Join
    SELECT *
    FROM person
    INNER JOIN poststed
    ON person.postnummer = poststed.postnummer
    WHERE person_id=(?)
```

Bcrypt

Get Bcrypt:

```
npm install bcrypt
```

Hash password:

```
// Import:
const bcrypt = require('bcrypt')

//Use in app.js
const hash = bcrypt.hashSync(password, 10);

// Export:
exports.bcrypt = bcrypt
```

↑ Put the above in their approtirate places↑

Check hashed password

```
// Use inside app.js
// Returns true or false
const check = bcrypt.compareSync(password, hash);
```

Example:

```
if(check){
 //Do something here if password is correct
}
```

Multer

Get Multer:

```
npm install multer
```

Setup of multer:

```
// Import:
const multer = require('multer')

//File upload set up
var storage = multer.diskStorage({
   destination: function (req, file, cb) {
     cb(null, './public/uploads') // create new folder in Public na-
med Uploads!!
   },
   filename: function (req, file, cb) {
     cb(null, file.originalname)
   }
})
const upload = multer({ storage: storage })

// Export:
exports.upload = upload
```

Examples

In the next pages there wil be shown some examples of different setups and projects.

The needed modules will be at the top, and setups of different modules can be found in the previous pages.

ps. Check database if db is needed

Forms

Needed modules: Express

Create a very simple form with post method

forms.hbs

```
  <form action="/formHandler" method="post">

      <!-- Add any input here -->
      <label for="name"> Name: </label>
      <input type="text" name="name">

      <!-- Submit form, button can also be used here -->
      <input type="submit" value="Submit">

   </form>
```

app.js

```
// Render the form page, This is not needed if file is .html
app.get('/form', (request,response) => {response.render("forms.hbs",
{})})

// this handles the form once submitted
app.post('/formHandler', (request,response) => {
   // to get the form content you can use request.body
   const name = request.body.name
   //redirect back to any desired page
   response.redirect('/form')});
```

Login

Needed modules: Express, Sessions, db, bcrypt

using the same login handler with query

login.hbs

```
    <legend>Login:</legend>
    <form action="/loginHandler?type=login" method="post">
        <!-- Login inputs -->
        <label for="name">Name</label>
        <input type="text" name="name">
        <label for="password">Password</label>
        <input type="password" name="password">
        <!-- Submit login-form, button can also be used here -->
        <input type="submit" value="Submit">
    </form>


    <legend>Create account:</legend>
    <form action="/loginHandler?type=create" method="post">
        <!-- Login inputs -->
        <label for="name">Name</label>
        <input type="text" name="name">
        <label for="password">Password</label>
        <input type="password" name="password">

        <input type="submit" value="Submit">
    </form>
```

app.js

```
// Render the login page, This is not needed if file is .html
app.get('/login', (request,response) => {response.render("login.hbs", {})})

// this handles the login-form once submitted
app.post('/loginHandler', (request,response) => {
    // to get the form content you can use request.body
    const name = request.body.name
    const password = request.body.password
    const type = request.query.type
    request.session.loggedIn = false
    if(type === 'login'){
        //Selecting with name
        const sql = db.prepare("SELECT * FROM login WHERE name = ?")
        const result = sql.get(name)
        // if result is true set loggedin to true and if so check if password is true
        if(result && (bcrypt.compareSync(password, result.password))){
            request.session.loggedIn = true
            //Redirect to any desired page
            response.redirect('/') }
        } else{response.redirect('/login')}
    if(type === 'create'){
        //Hasing the password
        const hash = bcrypt.hashSync(password, 10);
        //insert statement into the database file
        const sql = db.prepare("INSERT INTO login (name, password) VALUES (?,?)")
        const result = sql.run(name, hash)
        request.session.loggedIn = true
        //Redirect to any desired page
        response.redirect('/')
    }});
```

Display #each

Needed modules: Express, db, hbs

Here will be a display of items in the database file, using #each

app.js

```
app.get('/display', (request,response) => {
   // Get the items from the database
   const sql = db.prepare("SELECT * FROM display")
   const result = sql.all()
   response.render("display.hbs", {
      items:result
   }); });
```

display.hbs

```
  {{#each items}}
      <p>Index: {{@index}} </p>
      <p>Name: {{this.name}} </p>
      <p>Description: {{this.desc}} </p>
      <p>------------------------------------</p>
   {{/each}}
```

Shoppingcart

Needed modules: Express, db, hbs

This part will show how to add and remove from a shopping cart, as well reviewing it. we will be building on the display #each

shoppingcart.hbs

```
<h1>Add something to shoppingcart</h1>
   {{#each items}}
      <p>Index: {{@index}} </p>
      <p>Name: {{this.name}} </p>
      <p>Description: {{this.desc}} </p>
      <!-- Using query to send information -->
      <a href="/cartHandler?type=addToCart&id={{this.id}} ">add to cart</a>
      <p>------------------------------------</p>
    {{/each}}
   <h1>See the shoppingcart here</h1>
      {{#each shoppingcart}}
      <p>Index: {{@index}} </p>
      <p>Name: {{this.name}} </p>
      <p>Description: {{this.desc}} </p>
      <!-- Using query to send information -->
      <a href="/cartHandler?type=delFromCart&id={{this.id}} ">delete from cart</a>
      <p>------------------------------------</p>
    {{/each}}
   <!-- Here we will delete the whole order -->
   <a href="/cartHandler?type=delCart&id={{shoppingcart.[0].order_id}} ">delete
all</a>
```

app.js

```
app.get(’/shoppingcart’, (request,response) => {

   // Get the items from the database
   const sql = db.prepare("SELECT * FROM items")
   const result = sql.all()

   //Using inner join to get the shoppingcart, this can get complicated
   const sql2 = db.prepare(`
      SELECT orders_has_items.*, orders.status, items.name, items."desc"
      FROM orders_has_items
      JOIN orders ON orders_has_items.order_id = orders.id
      JOIN items ON orders_has_items.item_id = items.id
      WHERE orders.status = ’shoppingcart’;`)
   const result2 = sql2.all()
   response.render("shoppingcart.hbs", {
      items:result,
      shoppingcart: result2
   });
});

//This will be handeling adding, delete and delete all in shoppingcart
app.get(’/cartHandler’, (request,response) => {
   const id = request.query.id
   const type = request.query.type

   //Add to cart handler
   if(type === ’addToCart’){
      //check if there’s already a order with shoppingcart status
      const sql = db.prepare(’SELECT * FROM orders WHERE status = ?’)
```

```
      var result = sql.get(’shoppingcart’)
      if(!result){//if there was no shoppingcart then make new
         const sql = db.prepare(’INSERT INTO orders (status) VALUES (?)’)//Default
is shoppingcart
         result = sql.run(’shoppingcart’)
      }
      const sql2 = db.prepare(’INSERT INTO orders_has_items (item_id, order_id)
VALUES (?,?)’)
      const result2 = sql2.run(id, result.id)
   }

   //Delete item from cart handler
   if(type === ’delFromCart’){
      const sql = db.prepare(’DELETE FROM orders_has_items WHERE id = ?’)
      const result = sql.run(id)
   }

   //Delete everything from cart
   if(type === ’delCart’){
      const sql = db.prepare(’DELETE FROM orders_has_items WHERE order_id
=?’)
      const result = sql.run(Number(id))
      console.log(result)
   }
   response.redirect(’/shoppingcart’)
});
```

Access level

Needed modules: Express, db, hbs

Here there will be text rendered depending on the access level in the database.

accessLevel.hbs

```
  <!-- this page should render differently depending on the access level
-->
       <h2>Welcome to your dashboard!</h2>
       <p>Change access level in app.js</p>

<!-- CUTSOM HELPER! it can be found in setup.js -->
{{#ifAnyEqual accessLevel "Administrator" }}
   <p>You have access to Administrator features.</p>

   {{else ifAnyEqual accessLevel "Advanced"}} <!-- This works just like
else if -->
   <p>You have advanced access.</p>

   {{else ifAnyEqual accessLevel "Basic"}}
   <p>You have basic access.</p>

   {{else}}
   <p>You don’t have access to this page.</p>
{{/ifAnyEqual}}
```

setup.js

```
//a custom helper that takes multiple arguments and compares them
hbs.registerHelper(’ifAnyEqual’, function() {
  // Convert the arguments object to an array and extract the options obje-
ct
  const args = Array.from(arguments);
  const options = args.pop();

  // Compare each argument to the next argument
  for (let i = 0; i < args.length; i += 1) {
    if (args[i] === args[i + 1]) {
      // Render the contents of the {{#if}} block if any two arguments are
equal
      return options.fn(this);
    }
  }
  // Render the contents of the {{else}} block if no arguments are equal
  return options.inverse(this);});
```

app.js

```
//Also an example on conditional rendering
app.get(’/accesslevel’, (request,response) => {
   const sql = db.prepare(’SELECT * FROM accessLevel WHERE id = ?’)
   //Try changing between 1-3 and see what happens
   const result = sql.get(1)
   response.render("accessLevel.hbs", {
      accessLevel : result.accessLevel,
   });});
```

Option/Sort

Needed modules: Express, hbs

This is a way of sorting or using options to get a value

option_sort.hbs

```
   <!-- The change will be handled in client.js -->
  <select name="option" id="option" onchange="optionSort()">
    <option value="none" selected>-none-</option>
    {{#each type}}
      <option value="{{this.type}}"> {{this.type}} </option>
    {{/each}}
  </select>

  {{#each items}}
    <div id="{{this.type}}" class="item">
      <p>Name: {{this.name}}  </p>
      <p>Type {{this.type}}: </p>
      <p>--------------------------</p>
    </div>
  {{/each}}

  <script src="js/client.js"></script>
```

client.js

```
function optionSort() {
  //Selecting the option id and getting all divs with item calss
  const option = document.getElementById('option').value;
  const itemArray = Array.from(document.getElementsByClassNa-
me('item'))

  //Looping through the items and setting their display
  for (let i = 0; i < itemArray.length; i++) {
    const element = itemArray[i];
    if (option === "none" || element.id === option) {
      element.style.display = "block";
     } else {element.style.display = "none";}
  }
 }
```

app.js

```
app.get('/option_sort', (request,response) => {
  //Using inner join to get the type name
  const sql = db.prepare(`
    SELECT *
    FROM sort_items
    JOIN sort_type ON sort_items.type_id = sort_type.id
  `)
  const result = sql.all()

  //Select all types
  const sql2 = db.prepare(`SELECT * FROM sort_type`)
  const result2 = sql2.all()
  response.render("option_sort.hbs", {
    items: result,
    type : result2
  });});
```