LAB 3

❖ CONTENT

- Customize web layout/template
- Decorate web UI with Materialize CSS framework
- Implement features: filter, search, sort
- Form validation: client-side, server-side
- Install new packages: express-session, bcryptjs
- Authentication (login/logout + register)
- Password encryption

➢ DEMO PROJECT *(updated)*

- New collection: **users**
  - ***username, password***: used for authentication
  - ***role***: used for authorization
  - *Note: single collection, no relationship*

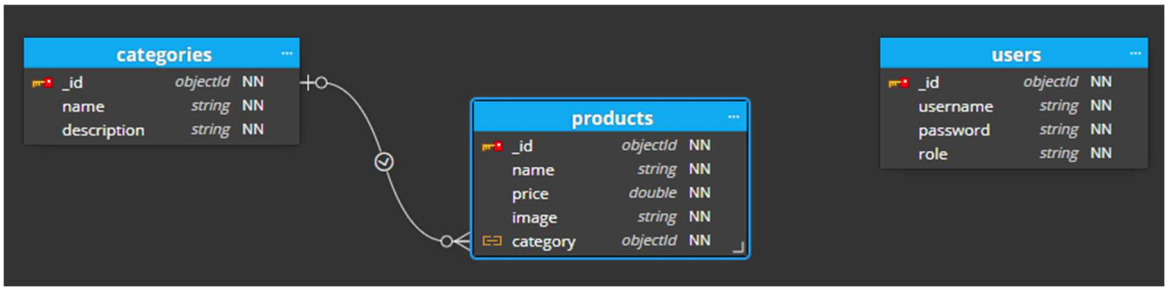


Figure 1 - Database diagram (updated)

❖ INSTRUCTION

1. Create new collection

*Figure 2 - Create collection "**users**" and populate data*

2. Customize web layout

- All web page use same web layout from **views/layout.hbs**
- To set custom (different) web layout, declare in route and create corresponding views
- Everything declared in web layout are similar for all pages such as CSS, JS, title, header, footer,… *except the main content* **{{{ body }}}**

*Figure 3 – Default web layout **(views/layout.hbs)***

```
router.get('/', function(req, res) {
  res.render('index', { layout: 'home_layout'});
});
```

*Figure 4 - Custom web layout **(views/home_layout.hbs)***

3. Decorate web UI with Materialize

Figure 5 - Copy CDN links from Materialize homepage to web layout (1)

```
<head>
  <title>{{title}}</title>
  <link rel='stylesheet' href='/stylesheets/style.css' />
  <!-- Compiled and minified CSS -->
  <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css">

  <!-- Compiled and minified JavaScript -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/js/materialize.min.js"></script>
</head>
```

Figure 6 – Copy CDN links from Materialize homepage to web layout (2)

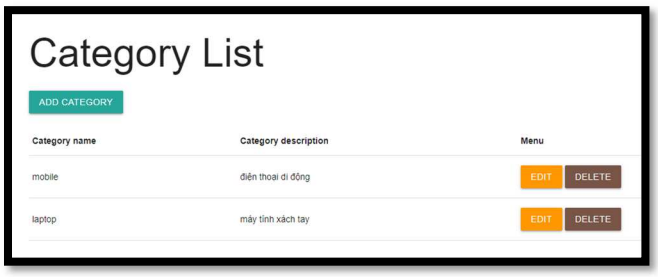
```
<script src="/scripts/materialize.js"></script>
```

*Figure 7 - Fix Materialize CSS framework drop-down bug (1) (**views/layout.css**)*

```
//fix Materialize drop-down bug
document.addEventListener('DOMContentLoaded', function () {
   var elems = document.querySelectorAll('select');
   var instances = M.FormSelect.init(elems);
});
```

*Figure 8 - Fix Materialize CSS framework drop-down bug (2) (**public/scripts/materialize.js**)*

```
<h1>Category List</h1>
<div class="row">
   <a class="btn" href="/category/add">Add category</a>
</div>
<table>
   <tr>
      <th>Category name</th>
      <th>Category description</th>
      <th>Menu</th>
   </tr>
   {{#each categoryList }}
   <tr>
      <td>{{ name }}</td>
      <td>{{ description }}</td>
      <td>
         <a class="btn orange" href="/category/edit/{{_id}}">Edit</a>
         <a class="btn brown" href="/category/delete/{{_id}}"
         onclick="return confirm('Do you want to delete this category ?')";
         >Delete</a>
      </td>
   </tr>
   {{/each}}
</table>
```




```
<form action="" method="post">
   <h4 class="center-align bold">Add new category</h4>
   <div class="row">
      <div class="input-field col s12 m4 offset-m4">
         <label for="">Category name</label>
         <input class="validate" type="text" name="name" id="">
      </div>
   </div>
   <div class="row">
      <div class="input-field col s12 m4 offset-m4">
         <label for="">Category description</label>
         <input class="validate" type="text" name="description" id="">
      </div>
   </div>
   <div class="row">
      <div class="col s12 m4 offset-m4 center-align">
         <input class="btn" type="submit" value="Add">
      </div>
   </div>
</form>
```

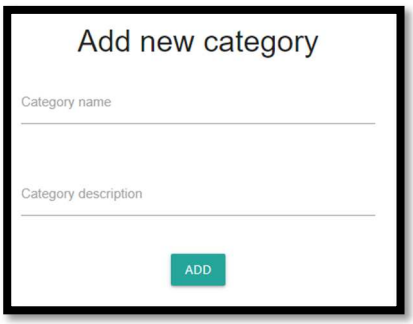


Figure 9 – Some web pages after decorating with Materialize

4. Implement filter feature (*ex: filter product by category*)

```
{{#each categoryList }}
<tr>
   <td>
      <a href="/category/detail/{{_id}}">
         {{ name }}
      </a>
   </td>
```

*Figure 10 – Front-end: **views/category/index.hbs***

```
router.get('/detail/:id', async (req, res) => {
   var id = req.params.id;
   var productList = await ProductModel.find({ category: id });
   res.render('product/index', { productList })
})
module.exports = router;
```

*Figure 11 – Back-end: **routes/category.js***

5. Implement search feature (*ex: search by product name*)

```
<div class="row">
   <form class="col m5" action="/product/search" method="post">
      <input class="form-control" type="search" name="keyword"
      placeholder="Search by product name" required>
   </form>
</div>
```

*Figure 12 - Front-end : **views/product/index.hbs***

```
router.post('/search', async (req, res) => {
   var keyword = req.body.keyword;
   var productList = await ProductModel.find({ name: new RegExp(keyword, "i") }).populate('category');
   res.render('product/index', { productList })
})
```

*Figure 13 - Back-end: **routes/product.js***

6. Implement sort feature (ex: *sort product name in asc or desc order*)

```
<div class="row">
   <a class="btn" href="/product/add">Add product</a>
   <a class="btn green" href="/product/sort/asc">Name ascending</a>
   <a class="btn lime" href="/product/sort/desc">Name descending</a>
</div>
```

*Figure 14 - Front-end : **views/product/product.hbs***

```
router.get('/sort/asc', async (req, res) => {
   var productList = await ProductModel.find().sort({ name: 1 }).populate('category');
   res.render('product/index', { productList })
})

router.get('/sort/desc', async (req, res) => {
   var productList = await ProductModel.find().sort({ name: -1 }).populate('category');
   res.render('product/index', { productList })
})
```

*Figure 15 - Back-end: **routes/product.js***

7. Form validation in client-side with HTML

```
<h1>Add new category</h1>
<input type="text" name="name" required
placeholder="Enter category name" minlength="3" maxlength="10">
```

*Figure 16 – Add HTML validation attributes to form (**views/category/add.hbs**)*

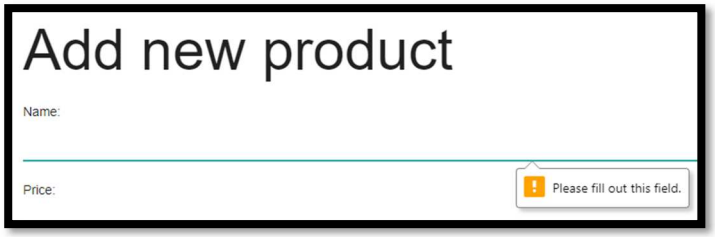


Figure 17 – Result of using HTML validation attributes

8. Form validation in client-side with JavaScript (JS)

```
<form action="/auth/register" method="post" onsubmit="return checkPass();">
   <h4 class="center-align bold">Customer Registration</h4>
   <div class="row">
      <div class="input-field col s12 m4 offset-m4">
         <label>Username</label>
         <input class="validate" type="text" name="username" required id="">
      </div>
   </div>
   <div class="row">
      <div class="input-field col s12 m4 offset-m4">
         <label>Password</label>
         <input class="validate" type="password" name="password" required id="password">
      </div>
   </div>
   <div class="row">
      <div class="input-field col s12 m4 offset-m4">
         <span id="retype_error" class="error"></span>
         <label>Retype Password</label>
         <input class="validate" type="password" name="retype" required id="retype">
      </div>
   </div>
```

*Figure 18 – **Set element id** and **run JS function** after form submit (**views/auth/register.hbs**)*

```
<link rel='stylesheet' href="/stylesheets/style.css" />

<script src="/scripts/validation.js"></script>
```

*Figure 19 - Attach related CSS and JS files (**views/layout.hbs**)*

```
function checkPass() {
   var password = document.getElementById('password').value;
   var retype = document.getElementById('retype').value;
   var error = document.getElementById('retype_error');
   if (retype != password) {
      error.innerHTML = "Retype password does not match. Check again !";
      return false; //prevent form submission
   }
   else {
      error.innerHTML = ""; //clear previous error
      return true; //allow form submission
   }
}
```

*Figure 20 – Create JS function to validate data (**public/scripts/validation.js**)*

9. Form validation in server-side with Mongoose validation

```
var ProductSchema = mongoose.Schema(
   {
      name: {
         type: String,
         minLength: [3, 'Product name must be at least 3 chracters'],
         maxLength: 30
      },
      price: {
         type: Number,
         required: true,
         min: [0, 'Product price can not be negative'],
         max: 1000
      },
```

*Figure 21 - Add validation rules in model (**models/ProductModel.js**)*

```
router.post('/add', async (req, res) => {
   try {
      var product = req.body;
      await ProductModel.create(product);
      res.redirect('/product');
   } catch (err) {
      if (err.name === 'ValidationError') {
         let InputErrors = {};
         for (let field in err.errors) {
            InputErrors[field] = err.errors[field].message;
         }
         res.render('product/add', { InputErrors, product });
      }
   }
})
```

*Figure 22 - Add **try ... catch** to return input error in route (**routes/product.js**)*

```
<h4 class="center-align bold">Add new product</h4>
<div class="row">
  <div class="input-field col s12 m4 offset-m4">
    <label for="">Product name</label>
    <input class="validate" type="text" name="name" value="{{ product.name }}">
    {{#if InputErrors.name}}
    <div class="error">
      {{InputErrors.name}}
    </div>
    {{/if}}
  </div>
</div>
```

*Figure 23 - Display error below form input (**views/product/add.hbs**)*

10. Install new packages

```
npm i express-session bcryptjs
```

Figure 24 - Install new packages (shorthand)

11. Authentication & password encryption

- Create new model **User**

```
var UserSchema = mongoose.Schema(
    {
        username: {
            type: String,
            unique: true
            required: true
        },
        password: {
            type: String,
            required: true
        },
        role: {
            type: String
        }
    }
);
var UserModel = mongoose.model('users', UserSchema);
```

*Figure 25 – Create **UserModel** (**models/UserModel.js**)*

- Import and config **express-session** package

```
//import "express-session" library
var session = require('express-session');
//set session timeout
const timeout = 10000 * 60 * 60 * 24; // 24 hours (in milliseconds)
//config session parameters
app.use(session({
  secret: "practice_makes_perfect", //your secret password
  saveUninitialized: false,
  cookie: { maxAge: timeout },
  resave: false
}));
```

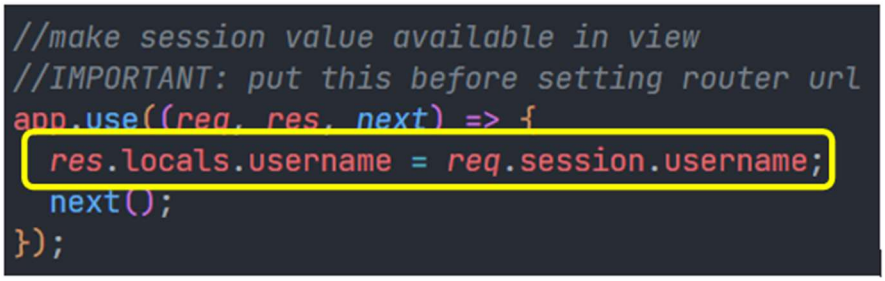


*Figure 26 - Config **express-session** package (**app.js**)*

- Declare and create new router **auth**

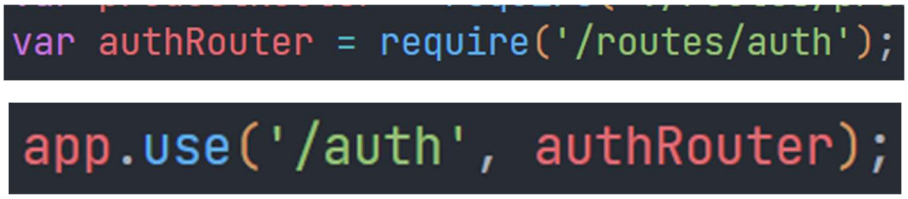


*Figure 27 - Declare **auth** router (**app.js**)*

*Figure 28 – Register new user and encrypt password with **bcryptjs** package (**routes/auth.js**)*

*Figure 29 – Login check and initialize session if success (**routes/auth.js**)*

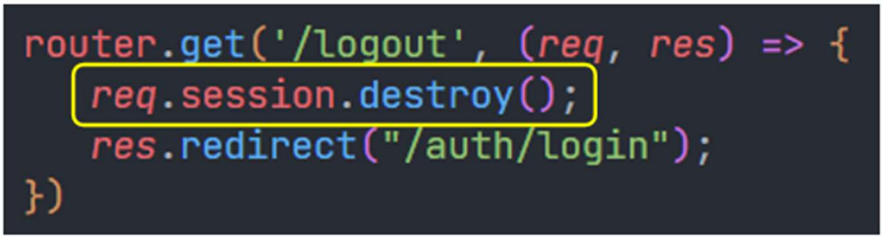


*Figure 30 - Logout by destroying session (**routes/auth.js**)*

- Create a middleware to validate user login session before they can access authorized web pages

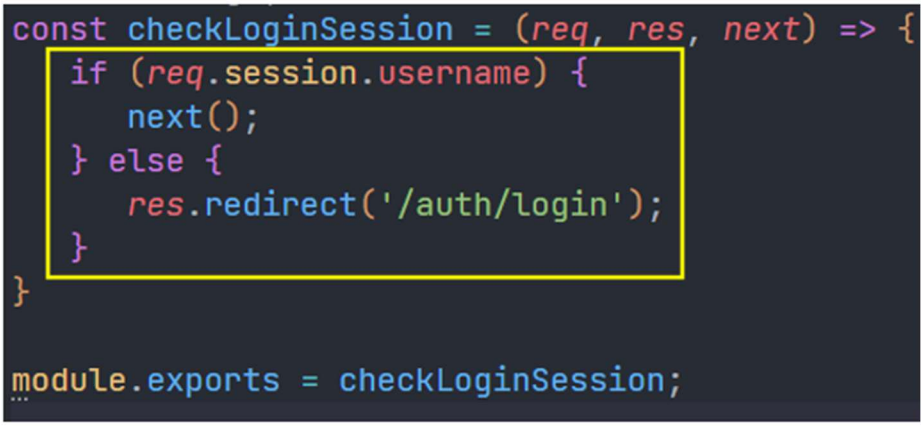


*Figure 31 - Create **auth** middleware to validate user login (**middlewares/auth.js**)*

```
const checkLoginSession = require('../middlewares/auth');

router.get('/', checkLoginSession function (req, res) {
  res.render('index', { layout: 'home_layout' });
});
```

*Figure 32 - Import and insert this middleware to route functions (**routes** folder)*

- Create all remained views to complete web app

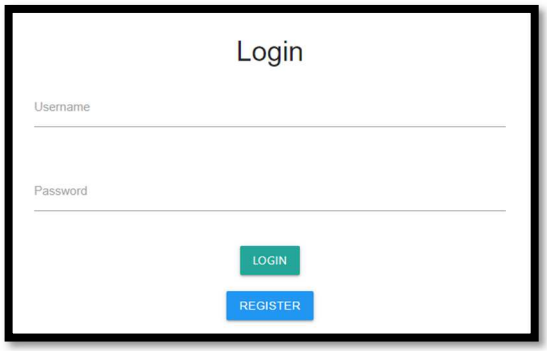
```
<center>
    <a href="/product">
        <img src="/images/product_management.jpg" alt="">
    </a>
    <h1>Welcome back, {{ username }} !</h1>
</center>
```

*Figure 33 - Homepage (**views/index.hbs**)*

```
<li>
  <a href="/auth/logout">LOGOUT ({{ username }})</a>
</li>
```

*Figure 34 - Insert **logou**t link to navigation bar (**views/layout.hbs**)*

```
<form action="/auth/login" method="post">
    <h4 class="center-align bold">Login</h4>
    <div class="row">
        <div class="input-field col s12 m4 offset-m4">
            <label>Username</label>
            <input class="validate" type="text" name="username" required id="">
        </div>
    </div>
    <div class="row">
        <div class="input-field col s12 m4 offset-m4">
            <label>Password</label>
            <input class="validate" type="password" name="password" required id="">
        </div>
    </div>
    <div class="row">
        <div class="col s12 m4 offset-m4 center-align">
            <input class="btn" type="submit" value="Login">
        </div>
    </div>
    <div class="row">
        <div class="col s12 m4 offset-m4 center-align">
            <a class="btn blue" href="/auth/register">Register</a>
        </div>
    </div>
</form>
```




```
<form action="/auth/register" method="post">
    <h4 class="center-align bold">Register</h4>
    <div class="row">
        <div class="input-field col s12 m4 offset-m4">
            <label>Username</label>
            <input class="validate" type="text" name="username" required id="">
        </div>
    </div>
    <div class="row">
        <div class="input-field col s12 m4 offset-m4">
            <label>Password</label>
            <input class="validate" type="password" name="password" required id="password">
        </div>
    </div>
    <div class="row">
        <div class="input-field col s12 m4 offset-m4">
            <label>Retype Password</label>
            <span class="retype_error"></span>
            <input class="validate" type="password" name="retype" required id="retype">
        </div>
    </div>
    <div class="row">
        <div class="col s12 m4 offset-m4 center-align">
            <input class="btn" type="submit" value="Register">
        </div>
    </div>
</form>
```

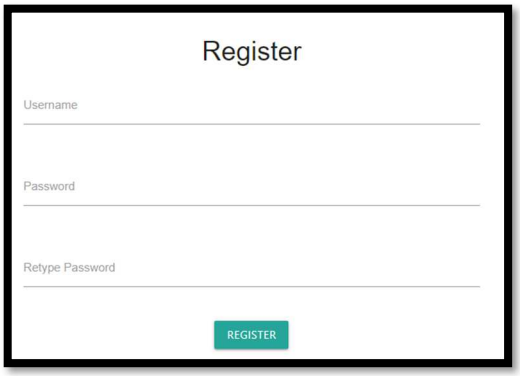


*Figure 35 - Login & register pages **(views/auth/login.hbs + register.hbs)***