# Penggunaan EJS

Pada modul ini, kita akan menggunakan EJS sebagai sebuah *template engine* untuk membuat sebuah *web application* menggunakan Node.js. Dokumentasi lengkap dari EJS dapat dipelajari pada https://ejs.co/.

Graphical user interface, website

Description automatically generated

Gambar 1. Embedded JavaScript templating atau EJS

## 1. Menyiapkan Project

Mula-mula, jalankan perintah di bawah ini melalui Terminal atau Command Prompt pada direktori tempat kode program anda disimpan untuk menginisialisasi NPM.

$ npm init

Setelah mengisi berbagai informasi mengenai project yang muncul pada saat anda menjalankan perintah di atas, jalankan perintah di bawah ini untuk instalasi *package* yang akan digunakan.

$ npm install –save express nodemon body-parser ejs

Express merupakan pustaka yang akan digunakan untuk membuat web server pada Node.js. Pustaka body-parser digunakan untuk melakukan *parsing* pada *body request* HTTP dan nodemon merupakan pustaka yang digunakan untuk mempermudah *testing* aplikasi, sehingga kita tidak perlu selalu mematikan dan menjalankan kembali perintah node index.js setiap kali membuat perubahan pada kode program. Tambahkan baris kode berikut ini pada package.json untuk menjalankan nodemon.

{

...

“scripts”: {

“dev”: “nodemon”,

...

}

Kemudian buatlah *file* index.js dan ketikkan kode program di bawah ini:

const express = require('express');

const bodyParser = require('body-parser');

const app = express();

// set the view engine to ejs

app.set('view engine', 'ejs');

// body-parser to parse request body

app.use(bodyParser.urlencoded());

// static files

app.use(express.static('public'));

// routes

// start server on port 3000

app.listen(3000);

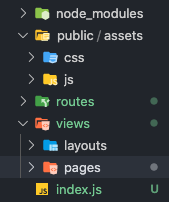
console.log('Server runs at port 3000...');

Jalankan nodemon pada Terminal atau Command Prompt untuk memastikan semua pustaka telah terinstall dengan baik.

$ npm run dev

Mula-mula buatlah beberapa folder pada direktori kode program anda, yaitu sebagai berikut:

* ./public
* ./public/assets
* ./public/assets/css
* ./public/assets/js
* ./routes
* ./views
* ./views/layouts
* ./views/pages



Gambar 2. Direktori project ini setelah menambahkan beberapa folder baru

Folder ./public/assets merupakan folder untuk menyimpan file CSS dan JS yang dapat digunakan pada tampilan website yang akan dibuat. Sementara itu, folder views akan menyimpan template EJS sebagai tampilan website. Lebih jauh tentang kenapa perlu ada pemisahan direktori ./views/pages dan ./views/partials akan dilihat pada pembahasan selanjutnya. Selain itu, kita juga akan memisahkan routing pada aplikasi ini dengan menempatkannya di direktori ./routes.

## 2. Membuat Tampilan Login

Pertama, kita akan membuat tampilan login dengan menggunakan EJS. Buatlah file baru dengan nama login.ejs yang disimpan pada direktori ./views/pages. Tambahkan kode program di bawah ini:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login</title>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">

</head>

<body class="bg-secondary">

<div class="container-md mt-3">

<div class="card">

<div class="card-body">

<h1 class="card-title">Member Login</h1>

<h5 class="card-subtitle mb-2 text-muted">Use your credentials to access this system.</h5>

<br>

<form action="/auth/login" method="post">

<div class="form-group">

<label for="username">Username</label>

<input type="text" class="form-control" name="username" required autofocus>

</div>

<div class="form-group">

<label for="password">Password</label>

<input type="password" class="form-control" name="password" required>

</div>

<input type="submit" class="btn btn-success" name="submit" value="Login">

</form>

</div>

</div>

</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7RsweNrtN/tE3MoK7ZeZDyx" crossorigin="anonymous"></script>

</body>

</html>

Pada kode di atas, kita menggunakan Bootstrap juga untuk mempercantik tampilan website yang dibuat.

Perhatikan kode program pada tag HTML <form> di atas. Pada kode tersebut,

<form action=”/auth/login” method=”post”>

kita akan melakukan login dengan mengirimkan request ke URL /auth/login menggunakan method POST. Ingat-ingat kembali materi pada pertemuan sebelumnya mengenai HTTP request methods, bahwa method POST biasanya digunakan untuk mengirimkan data input ke server, dalam hal ini adalah username dan password yang akan dicek untuk login.

Informasi username dan password pada form ini akan dikirimkan ke /auth/login pada saat user menekan tombol submit/login.

Untuk menampilkan tampilan login di atas, kita akan membuat router pada file auth.js di direktori ./routes, seperti di bawah ini:

const express = require('express');

const router = express.Router();

router.get('/login', async (req, res) => {

res.render('pages/login');

});

module.exports = router;

Kemudian, tambahkan kode program di bawah ini pada index.js untuk menghubungkan Express dengan kode yang menyatakan routes login di atas. Tambahkan kode ini di bawah comment routes pada index.js.

// routes

const auth = require('./routes/auth');

app.use('/auth', auth);

Setelah menambahkan kode di atas, coba buka <http://localhost:3000/auth/login> pada browser untuk melihat hasilnya seperti pada gambar 3.

Selanjutnya, kita akan menambahkan kode program untuk memvalidasi username dan password yang diketik oleh user.

Tambahkan kode di bawah ini pada file ./routes/auth.js

router.post('/login', async (req, res) => {

// get user input

const username = req.body.username;

const password = req.body.password;

if (username === "admin" && password === "admin") {

// TODO: implement sessions to check user is logged-in

// redirect to member area

res.redirect('/');

}

else {

// TODO: render the login page with error information

}

});

Graphical user interface, text, application

Description automatically generated

Gambar 3. Tampilan Login

Kode di atas menunjukkan bahwa blok pengecekan login dilakukan jika ada request ke URL /auth/login dengan menggunakan method POST. Untuk mempermudah contoh ini, username dan password yang valid adalah “admin”.

Pada kode di atas, kita akan di-redirect ke halaman lain jika login berhasil. Terdapat dua bagian yang belum diimplementasi, yaitu bagian pencatatan session jika login berhasil (sehingga user tidak perlu login lagi) dan bagian yang memunculkan pesan error jika username atau password yang diisi salah.

Kita akan melengkapi bagian pesan error tersebut terlebih dahulu. Tambahkan kode berikut ini:

// render the login page with error information

res.render('pages/login', { error: 'Wrong username or password.' });

Potongan kode tersebut menyatakan bahwa jika terdapat error, maka kita akan menampilkan kembali tampilan login, namun kali ini dengan menambahkan pesan error seperti di atas.

Pada pages/login.ejs, tambahkan kode untuk menampilkan pesan error tersebut (jika ada). Letakkan kode di bawah ini setelah kode yang menunjukkan button login <input type=”submit” ... >.

<% if (locals.error) { %>

<span class="text-danger pl-2"><%= error %></span>

<% } %>

Graphical user interface, text, application, email

Description automatically generated

Gambar 4. Tampilan error pada login

Untuk mengecek apakah proses login berhasil dan user di-redirect ke halaman lain, maka tambahkan kode berikut ini pada index.js.

// routes

const index = require('./routes/index');

const auth = require('./routes/auth');

app.use('/', index);

app.use('/auth', auth);

Lalu buatlah file ./routes/index.js untuk meng-handle request pada URL / seperti berikut ini:

const express = require('express');

const router = express.Router();

router.get('/', async (req, res) => {

res.send('ok');

});

module.exports = router;

Maka jika anda mencoba login, maka tampilan web akan dialihkan ke halaman index / yang menampilkan teks “ok”.

## 3. Session

Langkah selanjutnya adalah menambahkan session untuk mencatat user yang sudah pernah login atau belum. Jika user belum login, maka akan kita arahkan ke halaman login. Namun sebaliknya, jika user sudah login, maka ia tidak dapat mengakses halaman login lagi (alihkan ke halaman index).

Mula-mula, install pustaka yang bernama express-session dengan menggunakan NPM.

$ npm install –save express-session

Tambahkan kode yang ditebalkan seperti di bawah ini pada index.js.

const express = require('express');

const bodyParser = require('body-parser');

**const session = require('express-session');**

const app = express();

// set the view engine to ejs

app.set('view engine', 'ejs');

// body-parser to parse request body

app.use(bodyParser.urlencoded());

// static files

app.use(express.static('public'));

**// enabling session**

**app.use(session({**

**secret: 'some\_secret\_key',**

**cookie: {}**

**}));**

// routes

const index = require('./routes/index');

const auth = require('./routes/auth');

app.use('/', index);

app.use('/auth', auth);

app.listen(3000);

console.log('Server runs at port 3000...');

Nilai some\_secret\_key disarankan untuk diisi dengan sebuah string acak yang digunakan sebagai key untuk mengenkripsi session yang dibuat. Kemudian, tambahkan kode berikut pada ./routes/auth.js pada bagian TODO untuk menyimpan session.

// implement sessions to check user is logged-in

req.session.user = "admin";

Pada contoh ini, kita hanya menyimpan string “admin” yang menunjukkan bahwa user telah login. Pada implementasi lain, informasi yang disimpan bisa saja ID user (misalnya ID yang menjadi PK pada table user di database) untuk mengindentifikasi user yang mengakses web ini.

Perlu diingat bahwa nilai yang disimpan session ini sendiri berada pada server, sehingga tidak dapat diakses oleh client atau browser. Untuk sistem penyimpanan pada client, anda dapat menggunakan cookies.

Tambahkan kode yang ditebalkan di bawah ini pada ./routes/index.js yang merupakan kode untuk mengecek apakah user sudah melakukan login atau belum dengan mengecek apakah session user telah tersimpan di server atau belum.

router.get('/', async (req, res) => {

**// check user session**

**if (!req.session.user) {**

**res.redirect('/auth/login');**

**} else {**

**res.send('ok');**

**}**

});

Anda dapat mencoba dengan langsung membuka <http://localhost:3000> tanpa melalui login sebelumnya. Anda akan dialihkan ke halaman login secara otomatis dikarenakan session belum ada. Jika anda telah melakukan login, maka halaman / akan menampilkan kata “ok”.

Tambahkan juga kode yang ditebalkan berikut pada ./routes/auth.js untuk mengalihkan user yang sudah login dari halaman login.

router.get('/login', async (req, res) => {

**// if the user has logged-in, redirect to index**

**if (req.session.user) {**

**res.redirect('/');**

**} else {**

**res.render('pages/login');**

**}**

});

## 4. Logout

Untuk melakukan logout, sederhananya adalah kita hanya perlu menghapus session yang dimiliki user yang sedang login. Caranya adalah dengan memanggil req.session.destroy() seperti pada contoh di bawah ini (./routes/auth.js):

router.get('/logout', async (req, res) => {

// destroy all session

req.session.destroy();

// redirect to login

res.redirect('/auth/login');

});

## 5. Layouts

Pada saat membuat tampilan web, seringkali tampilan-tampilan dari masing-masing halaman memiliki layout yang sama. Misalkan sebuah aplikasi reservasi hotel, tentunya memiliki layout tampilan yang sama untuk halaman depan, halaman check-in, halaman pencarian hotel, dan seterusnya.

Untuk mempermudah pembuatan web, maka kita dapat membuat sebuah layout dan kemudian cukup melengkapi bagian-bagian yang berbeda dari tampilan layout untuk halaman-halaman tertentu. Hal ini dapat dilakukan dengan bantuan pustaka express-ejs-layouts.

$ npm install –save express-ejs-layouts

Kemudian, tambahkan kode berikut pada index.js.

const layouts = require('express-ejs-layouts');

// use layouts

app.use(layouts);

app.set('layout', 'layouts/main.ejs');

// place all styles block in the layout at the head

app.set("layout extractStyles", true)

// place all scripts block in the layout at the end

app.set("layout extractScripts", true)

Buatlah file main.ejs pada direktori ./views/layouts yang merupakan layout utama dari halaman web kita.

Akan tetapi, ada beberapa halaman yang mungkin saja menggunakan layout yang berbeda, atau bahkan tidak menggunakan layout sama sekali. Contohnya pada project kita adalah halaman login.

Tambahkanlah penunjuk layout: false pada beberapa bagian di ./routes/auth.js seperti di bawah ini:

router.get('/login', async (req, res) => {

if (req.session.user) {

res.redirect('/');

} else {

**res.render('pages/login', { layout: false });**

}

});

router.post('/login', async (req, res) => {

// get user input

const username = req.body.username;

const password = req.body.password;

if (username === "admin" && password === "admin") {

// implement sessions to check user is logged-in

req.session.user = "admin";

// redirect to member area

res.redirect('/');

}

else {

// render the login page with error information

**res.render('pages/login', { layout: false, error: 'Wrong username or password.' });**

}

});

Kemudian, kita akan membuat contoh layout pada ./views/layouts/main.ejs sebagai berikut:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title**><%- title %>**</title>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css" integrity="sha384-TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">

**<%- style %>**

</head>

<body>

<div class="container-md mt-3">

**<%- body %>**

</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7RsweNrtN/tE3MoK7ZeZDyx" crossorigin="anonymous"></script>

**<%- script %>**

</body>

</html>

Perhatikan beberapa bagian yang ditebalkan pada kode program di atas. Bagian tersebut merupakan bagian yang akan diisi sesuai dengan halaman web yang ditampilkan.

Misalkan pada <%- title %> dapat diisi sesuai judul halaman yang akan ditampilkan. Khusus untuk <%- style %> dan <%- script %> merupakan bagian yang akan diisi dengan script JS dan CSS secara otomatis karena kita sudah mengeset konfigurasi extractStyles dan extractScripts di index.js.

Buatlah file ./views/pages/index.js dan isi dengan kode program di bawah ini:

<h1>Member Area</h1>

<%- contentFor('title') %>

Member Area

Dengan menggunakan layouts, maka tulisan heading 1 di atas akan dimunculkan pada bagian <%- body %> dan tulisan setelah <%- contentFor(‘title’) %> dimunculkan di bagian <%- title %> sesuai dengan layout yang telah dibuat.

## 6. NavBars

Kita akan melanjutkan contoh dengan membuat top navbar pada layout di atas.

Tambahkan kode HTML di bawah ini pada ./views/layouts/main.ejs setelah <body> namun sebelum <div class=”container-md mt-3”>.

<nav class="navbar navbar-expand-lg navbar-light bg-light">

<a class="navbar-brand" href="#">Navbar</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav mr-auto">

<li class="nav-item">

<a class="nav-link" href="/">Home</a>

</li>

<li class="nav-item">

<a class="nav-link" href="/todo">To-Do List</a>

</li>

</ul>

<ul class="navbar-nav">

<li class="nav-item">

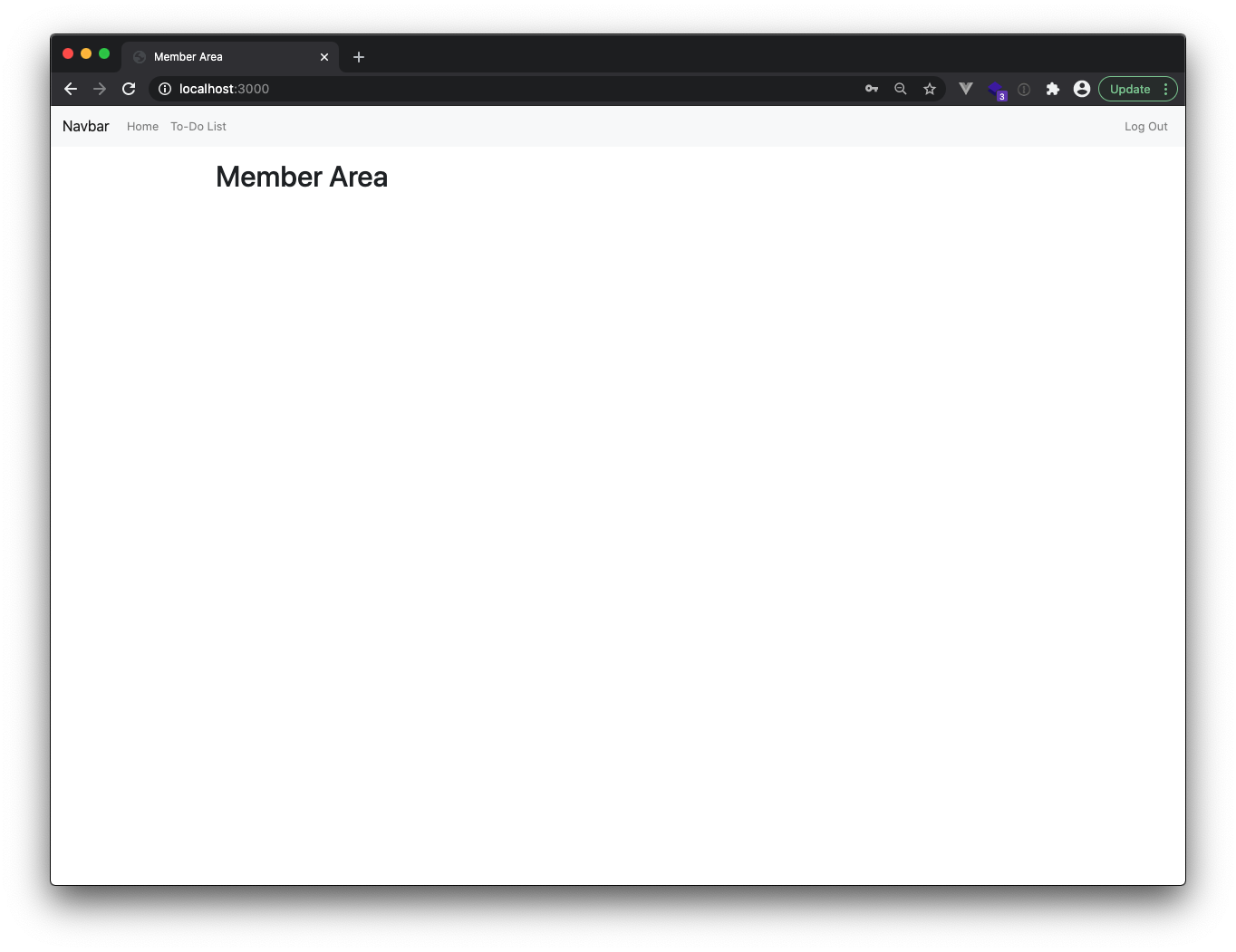
<a class="nav-link" href="/auth/logout">Log Out</a>

</li>

</ul>

</div>

</nav>



Gambar 5. Tampilan index setelah ditambahkan navbar

## 7. To-dos

Pada navbar terdapat menu To-Dos yang akan kita buat sekarang. Fitur to-dos ini seperti task management atau to-do lists yang biasa digunakan, yaitu untuk menambahkan task atau menghapus task yang sudah selesai dikerjakan.

Mulailah dengan menambahkan kode berikut ini pada index.js pada bagian routes.

const todo = require('./routes/todo');

app.use('/todo', todo);

Lalu, buatlah file ./routes/todo.js dan isi dengan kode program di bawah ini:

const express = require('express');

const router = express.Router();

router.get('/', async (req, res) => {

res.render('pages/todo');

});

module.exports = router

Dari kode di atas, kita perlu membuat file ./views/pages/todo.ejs yang akan ditampilkan setiap kali user membuka <http://localhost:3000/todo>.

<h1>To-Do List</h1>

<div class="card">

<div class="card-body">

<form action="/todo/add" method="post">

<div class="form-group">

<label for="taskName">Insert a new task</label>

<input type="text" class="form-control" name="taskName" required autofocus>

</div>

<input type="submit" class="btn btn-primary" name="submit" value="Save">

</form>

</div>

</div>

<br>

<script src="https://unpkg.com/feather-icons"></script>

<script>feather.replace()</script>

<%- contentFor('title') %>

To-Do List

Kode program di atas akan menampilkan form dimana user dapat mengetikkan nama task dan menyimpannya. Untuk menyimpan task yang ditulis oleh user, maka kita perlu membuat POST ke /todo/add seperti pada form di atas.

Penyimpanan tasks tersebut akan dilakukan menggunakan session. Perhatikan kode di bawah ini yang dituliskan pada ./routes/todo.js.

router.post('/add', async (req, res) => {

// if there's no tasks in the session, create one

if (!req.session.tasks) {

req.session.tasks = [];

}

// add new task

const newTask = req.body.taskName;

req.session.tasks.push(newTask);

res.redirect('/todo');

});

Perintah if di awal adalah untuk mengecek jika session tasks belum pernah dibuat sebelumnya, maka inisialisasi dengan array kosong. Kemudian, task baru tersebut dimasukkan ke array tersebut.

Untuk menampilkan tasks apa saja yang sudah tersimpan, pertama-tama, ubah kode pada ./routes/todo.js pada bagian GET / seperti di bawah ini:

router.get('/', async (req, res) => {

res.render('pages/todo', { tasks: req.session.tasks });

});

Kemudian, tambahkan ke ./views/pages/todo.ejs

<h1>To-Do List</h1>

<div class="card">

<div class="card-body">

<form action="/todo/add" method="post">

<div class="form-group">

<label for="taskName">Insert a new task</label>

<input type="text" class="form-control" name="taskName" required autofocus>

</div>

<input type="submit" class="btn btn-primary" name="submit" value="Save">

</form>

</div>

</div>

<br>

**<table class="table">**

**<thead>**

**<tr>**

**<th>#</th>**

**<th>Task Name</th>**

**<th></th>**

**</tr>**

**<% if (!locals.tasks || tasks.length == 0) { %>**

**<tr><td colspan="4" class="text-center"><i>No tasks! You're good to go.</i></td></tr>**

**<% } else { %>**

**<% tasks.forEach((task, index) => { %>**

**<tr>**

**<td><%= index + 1 %></td>**

**<td><%= task %></td>**

**<td>**

**<form action="/todo/done/<%= index %>" method="post">**

**<button type="submit" class="btn btn-outline-success btn-sm">**

**<i data-feather="check"></i>**

**Done**

**</button>**

**</form>**

**</td>**

**</tr>**

**<% }) %>**

**<% } %>**

**</thead>**

**</table>**

<script src="https://unpkg.com/feather-icons"></script>

<script>feather.replace()</script>

<%- contentFor('title') %>

To-Do List

Jika variabel tasks yang disimpan pada session tidak pernah dibuat atau isinya kosong, maka pesan tidak ada tasks akan ditampilkan. Jika terisi, maka dengan perintah forEach dilakukan looping untuk setiap isi array tasks dan menampilkan isi dari task tersebut.

Selain itu, pada kode di atas dibuat juga sebuah button untuk menghapus task yang sudah selesai. Button tersebut akan memanggil POST /todo/done/<index> dengan <index> merupakan lokasi task yang mau dihapus di dalam array.

Tambahkan kode di bawah ini pada ./routes/todo.js untuk meng-handle penghapusan tasks.

router.post('/done/:index', async (req, res) => {

// get the index of the task to be deleted

const index = req.params.index

// only delete if there's that task

if (req.session.tasks && index < req.session.tasks.length) {

req.session.tasks.splice(index, 1);

}

res.redirect('/todo');

});

Graphical user interface, application

Description automatically generated

Gambar 6. Tampilan To-Do List