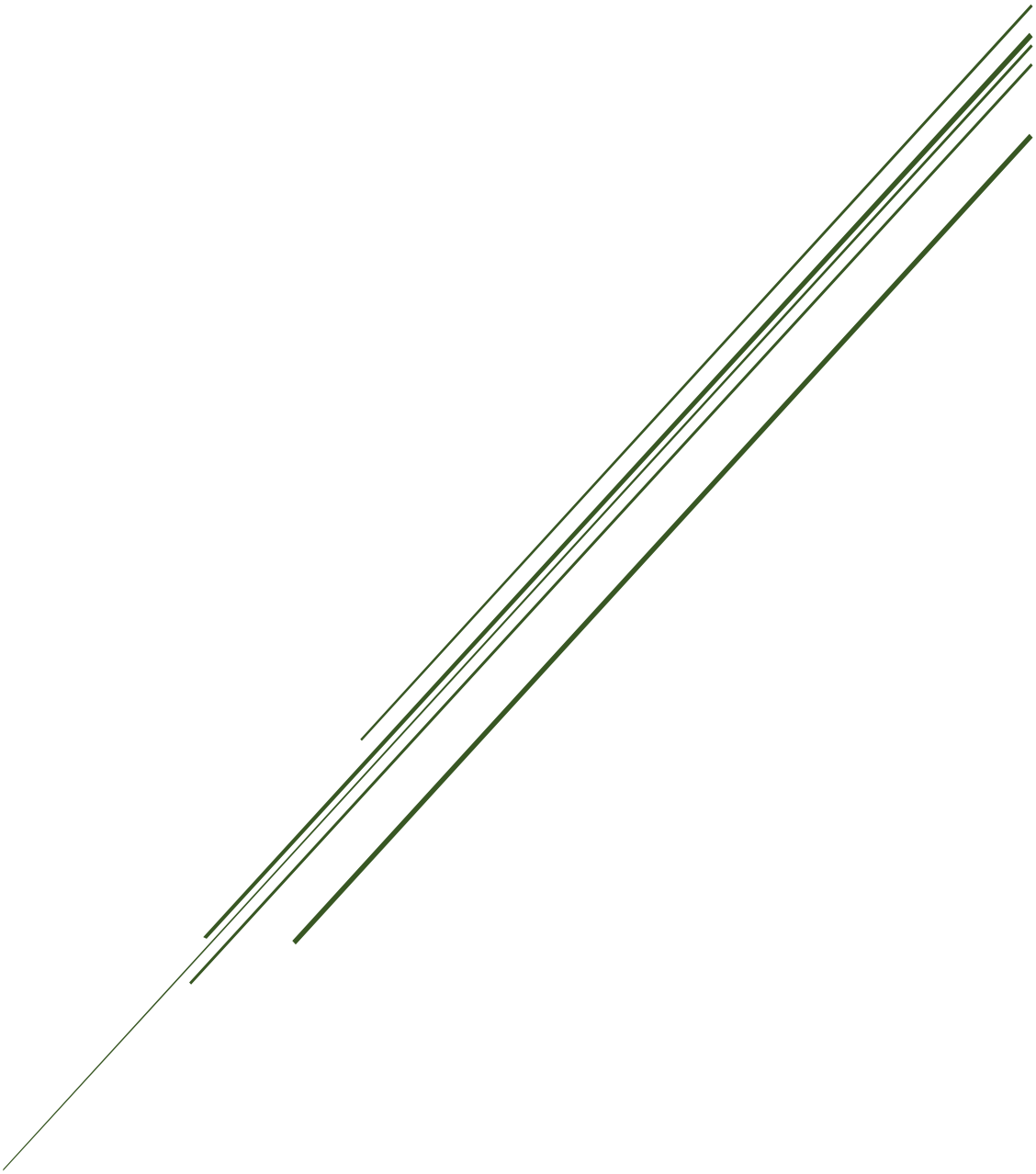
WEB APP USING NODE.JS WORKSHOP

BUILDING A BASIC TODO LIST APP



Organized by DSC UUM
Prepared by Ilham Muslim M.R. (ilham.mmr@gmail.com)

Table of Contents

INSTALLATION

NODE JS

Download link : https://nodejs.org/en/download/

Nodejs installation on windows 10 tutorial video : https://www.youtube.com/watch?v=8xRfg-oy16U

MongoDB

Download link : https://www.mongodb.com/try/download/community

Mongodb installation on windows 10 tutorial video:
https://www.youtube.com/watch?v=TMdRuhCzvKs

Visual Studio Code

Download Link : https://code.visualstudio.com/

TODOWEBAPP-STARTER

This is the starter file for the workshop.

Github link : https://github.com/ilham-mmr/WebApp-Using-Nodejs-workshop

WHAT IS WEB APP

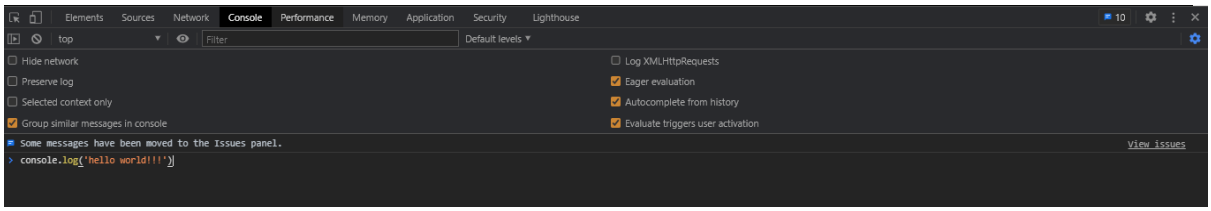
A web application is a software program that runs on a web server. Unlike traditional desktop applications launched by your operating system, web apps must be accessed through a web browser.

INTRODUCTION TO NODE JS

Before we dive into node js, I will introduce you to Javascript. Javascript differs from the Java programming language that is studied by most early-semester computer science or IT students. Javascript is a programming language that is used for writing scripts on the website to make it interactive. For instance, you use javascript to show or hide more information with the click of a button. So initially, javascript could only be run in the browser.

The following is how javascript is executed on a browser like chrome. Open the browser and create a program that says hello world in the browser.

Press Command+Option+C (Mac) or Control+Shift+C (Windows, Linux, Chrome OS) to open chrome dev tools. Then, type console.log('hello world!')



It will spit out hello world

```
> console.log('hello world!!!')
  hello world!!!
```

.

However, before nodejs came into existence, there was no way of running javascript code outside the browser. In 2009 Ryan Dahl took advantage of Chrome's V8 JavaScript engine and made it available for running javascript outside the browser. Besides, Nodejs can now be used for server-side. Hence, you can use the same programming language for both the front-end and back end.

In addition, node js is a runtime environment based on the V8 engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

Let us create a Hello world program in nodejs as follows :

1. open vscode

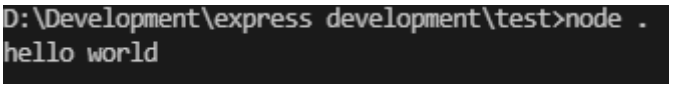
2. create a file name index.js

3. write the code below :

```
console.log("hello world");
```

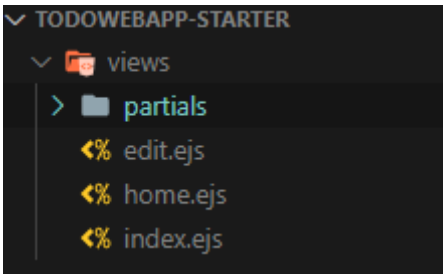
Open your favorite terminal and type *node .* to run the code.

Output:



INITIALIZE A NODE JS PROJECT

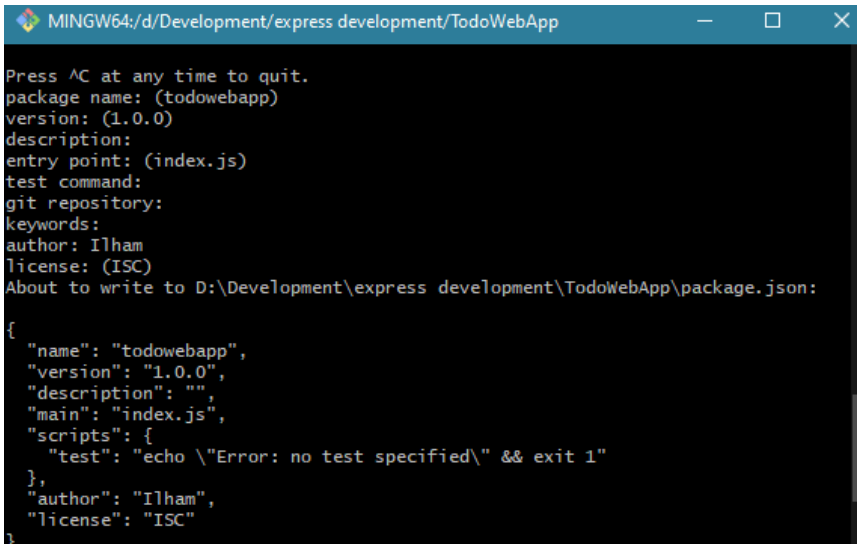
Firstly, you need to open the downloaded folder named **TodoWebApp-Starter.** It consists of a views folder that we will use later.



Open the folder using vscode and bring up a terminal. You can use the command prompt or gitbash, whichever that you like. In this case, I will use git bash.

To initialize a node js project, type *npm init*

You'll be asked to fill in some stuff. You can leave it blank to make it default by hitting enter, but you can fill in your name for the author.



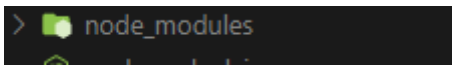
Now open vscode and open the project directory that we just created. You should see package.json. It consists of information that we just entered before.

RUN OUR FIRST SERVER

Creating our own server using pure node.js can be a hassle. We will use a popular web framework of node js, which is Express. To install that, we can simply type npm install express on our gitbash.

```
Ilham@LAPTOP-0B3A56T8 MINGW64 /d/Development/express development/TodoWebApp
$ npm install express
```

It will wait for a few seconds; basically, it will download the express web framework, and it will exist under the node_modules folder, which is created automatically for us.



Just ignore the folder. We will not worry about that.

Now create an app.js file that will write most of our code there. You can make it manually on vscode, or you can use the git bash command touch app.js.

```
Ilham@LAPTOP-0B3A56T8 MINGW64 /d/Development/express development/TodoWebApp
$ touch app.js
```

You should see the app.js file in the project directory. Now open the file and create our first server. Type this code

```
const express = require('express');
const app = express();


app.listen(3000, function(){
    console.log('server created');
});
```

To run our server, we need to open our git bash again and type ***node app.js***

```
Ilham@LAPTOP-0B3A56T8 MINGW64 /d/Development/express development/TodoWebApp
$ node app.js
server created
```

Now open the browser. Type **localhost:3000**


```
Cannot GET /
```

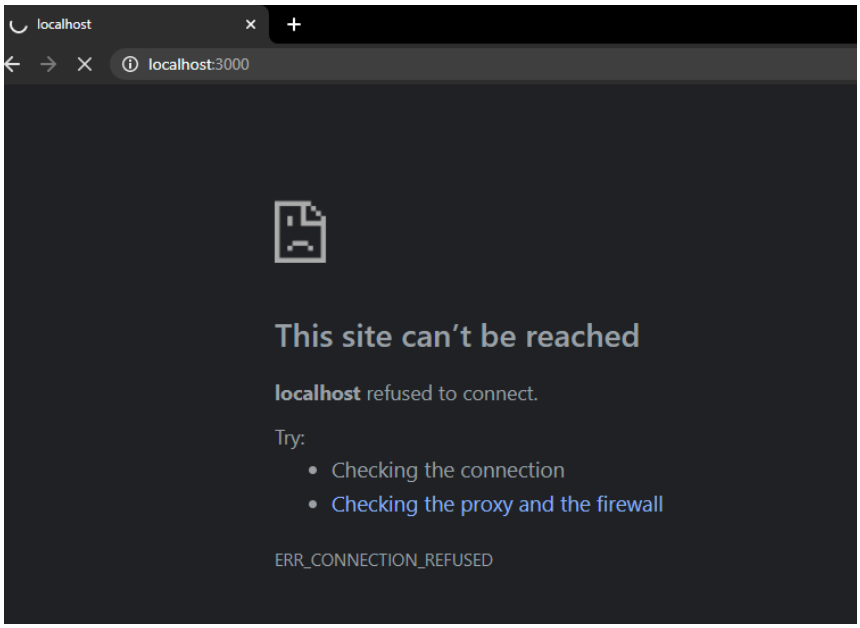
If you see the "Cannot GET /", don't worry. We will fix that later when we learn about routings. We don't have any routes yet for our app.

To stop our server from the git bash, hit CTRL + C.You should go back to $ prompt like in the below image.

```
Ilham@LAPTOP-0B3A56T8 MINGW64 /d/Development/express development/TodoWebApp
$ node app.js
server created

Ilham@LAPTOP-0B3A56T8 MINGW64 /d/Development/express development/TodoWebApp
$
```

If we try to reaccess the localhost:3000, we can't access it now since we just turned our server off.



ADDING GLOBAL PACKAGES

Instead of typing node index.js repeatedly to run the server, we will instead install nodemon globally. Nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.

To install nodemon simply type : npm install -g nodemon

GETTING TO KNOW WITH MONGO DB AS DATABASE

Install mongoose with npm install mongoose

```
Ilham@LAPTOP-0B3A56T8 MINGW64 /d/Development/express development/TodoWebApp
$ npm install mongoose
```

First thing first, we need to define our schema. Create a model directory and create a todos.js file.

Type the below code into todos.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const TodoSchema = new Schema({
    name: String,
});

module.exports = mongoose.model('Todo', TodoSchema);
```

Let's incorporate it into the app.js file. However, we need to connect it to the MongoDB database first.

https://mongoosejs.com/docs/index.html

1. we need to require mongoose first

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/todoapp',
    {
        useNewUrlParser: true,
        useUnifiedTopology: true
    });
```

We have a pending connection to the test database running on localhost. We now need to get notified if we connect successfully or if a connection error occurs:

```
const db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function () {
    console.log('database connected');
});
```

After we write the above code. We also need to import our Todo model.

```
const Todo = require('./model/todos');
```

now try to add a new todo item into our database.

```
const addTodo = new Todo({name:'learn web development at 5 pm'});
addTodo.save();
```

now run our server using node app.js . If done, we comment out or simply delete the above code because we want only to check whether we can save to the database or not

the commented code looks like this, and it will not be executed.

```
// const addTodo = new Todo({name:'learn web development at 5 pm'});
// addTodo.save();
```

If there's no error, it can successfully be saved in our database. To check that,

1. type mongo command. You'll be brought into the mongo terminal

```
$ mongo
MongoDB shell version v4.4.3
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("62aff028-eaf9-4ff4-a96e-ae3178056cd2") }
MongoDB server version: 4.4.3
---
The server generated these startup warnings when booting:
        2021-03-04T17:22:14.937+07:00: Access control is not enabled for the database. Read and write access to data and configuration
 is unrestricted
---
```

2. and then type show dbs to see the database in our MongoDB. Our database name is todoapp.

```
todoapp         0.000GB
workshop        0.000GB
```

3. to switch to the todoapp database. Type use todoapp

```
> use todoapp
switched to db todoapp
```

4. Now, to see a collection in todoapp database, type show collections You'll see todos

```
> show collections
todos
```

5. now, to see what we just saved into our database, write db.todos.find() to show our saved todo item.

```
> db.todos.find()
{ "_id" : ObjectId("60479e71dd252562789f38e0"), "name" : "learn web development at 5 pm", "__v" : 0 }
```

Now, if you see, there's _id. Mongo automatically assigns every document with a unique id.

5. To exit from mongo type exit and the mongo will say *bye.*

```
> exit
bye
```

TEMPLATING ENGINE USING EJS

EJS is a simple templating language that lets you generate HTML markup with plain JavaScript. Basically, it enables you to render the HTML page dynamically that you view on the browser. Ejs takes data, and ejs will render it. An example will be showing data from our database.

Inside of ejs file is a bunch of HTML but with some javascript code. That's why ejs stands for embedded javascript.

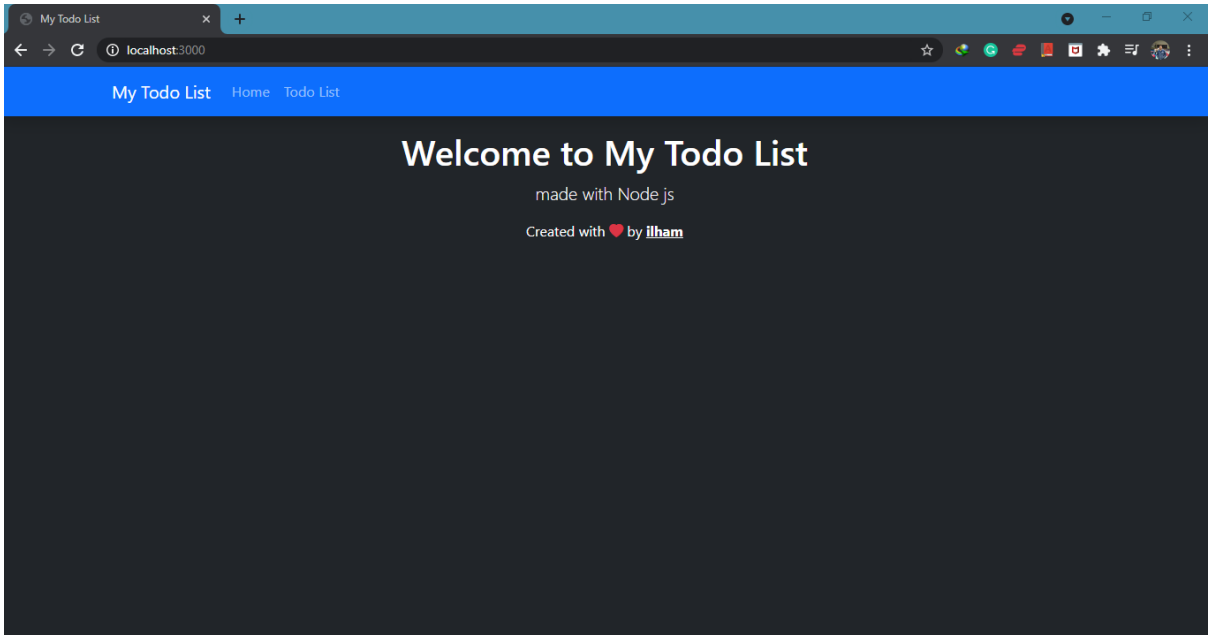
I have provided you with some templates that you can use, so you don't need to worry about the front-end stuff or our website's look.

To install ejs

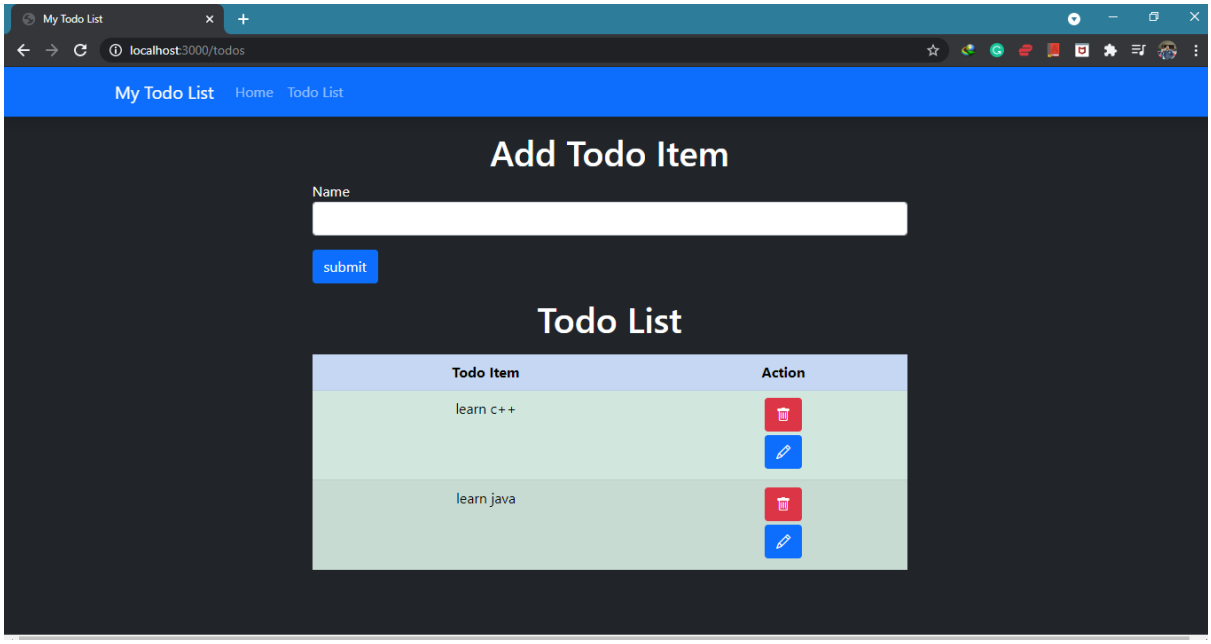
```
Ilham@LAPTOP-0B3A56T8 MINGW64 /d/Development/express development/TodoWebApp
$ npm install ejs
```

There are 3 pages for our website.

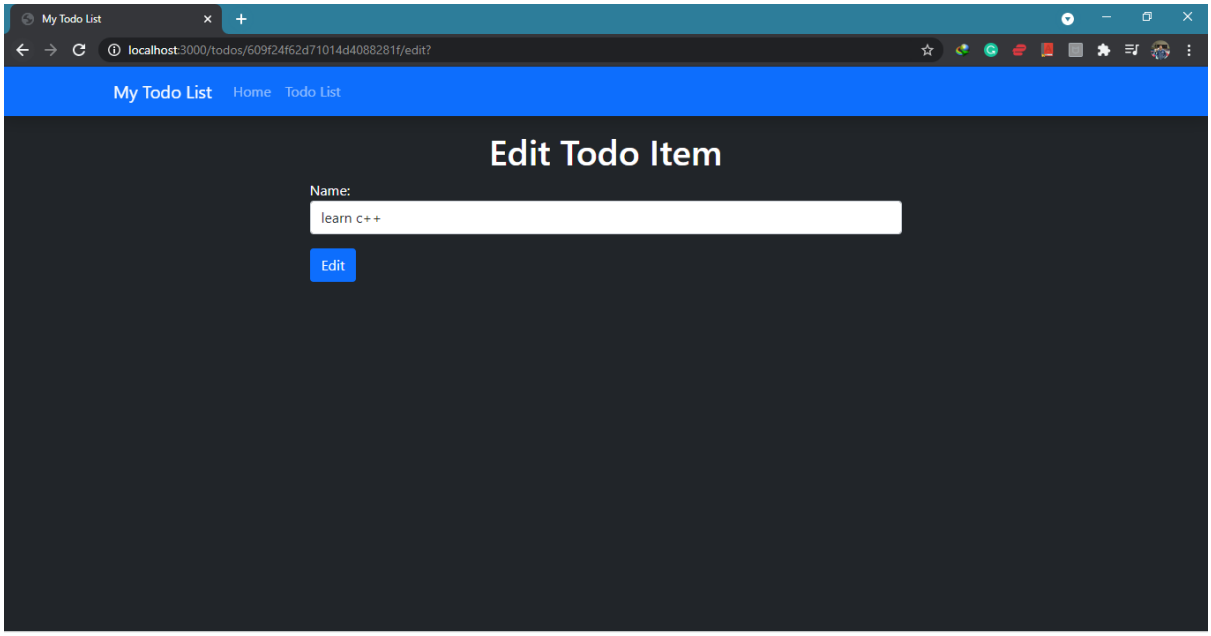
1. home.ejs page, which is the first page when a user visits the website.



2. index.ejs . this page displays our todo list and add item field



3. edit.ejs this page is used to edit our existing todo item.


My Todo List
localhost:3000/todos/609f24f62d71014d4088281f/edit?
My Todo List
Home
Todo List
Edit Todo Item
Name:
learn c++
Edit

REQUEST AND RESPONSE CYCLE

The request/response cycle traces how a user's request flows through the app. If a user entered facebook.com into their web browser, the web browser would request the server to look for the Twitter resource. Then a server sends an HTTP response to a client in response to an HTTP request.

OUR FIRST BASIC ROUTING

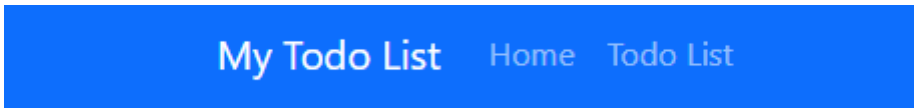
In this part, we will learn how to integrate the three pages we have into our server to access.

Firstly, we need to set our template engine. In this case, we use ejs in our app.

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

now we come to the exciting part.

Routing refers to determining how an application responds to a client request to a particular endpoint, a URI (or path) and a specific HTTP request method (GET, POST, and so on).




```
<div class="navbar-nav">
    <a class="nav-link" href="/">Home</a>
    <a class="nav-link" href="/todos ">Todo List</a>
</div>
```

The href attribute that's part of navbar specifies the URL of the page the link goes to.

Now we will match those links. When we hit that route, our server will render the requested page.

1. GET /

```
app.get('/', function  (req,res){
    res.render('home');
});
```

2. GET /todos

```
app.get('/todos', functio(req,res){
    res.render('index');
});
```

NEXT LEVEL OF ROUTING

In this part, we will add five more paths. We also will work with ejs and how to render data dynamically. Before we go doing that, I would like to introduce you to REST and HTTP verbs.

REST stands for Representational state transfer, a popular standard used by web applications that apply the CRUD pattern (Create, Read, Update and Delete) to create the requests and URIs of the application.

Instead of having a URL like site.com/add_todo, we just expose site.com/todos then we use the following different HTTP verbs to execute that resource.

HTTP VERBS

1.GET

To retrieve data from the server

2. POST

To submit data to the server

3. PUT

To update data entirely on the server

4. DELETE

To delete data from the server

ADDING OTHER ROUTES

1. Adding Todo Item

Before we use the post method. We need to add the below code.

```
app.use(express.urlencoded({ extended: true }));
```

according to documentation. It parses incoming request bodies in a middleware before your handlers, available under the req body property.

Without the above body-parser, our node js app doesn't know the body. The following error is the cause of not using URL-encoded

```
TypeError: Cannot destructure property 'name' of 'req.body' as it is undefined.
```

The encoding type of a form is determined by the attribute enctype application/x-www-form-urlencoded - Represents a URL encoded form. This is the default value if enctype attribute is not set to anything.

POST /todos

We will access req.body object. It will contain key-value pairs, where the value can be a string or array (when extended is false), or any type (when extended is true).

```
app.post('/todos', function (req, res) {
    const { name } = req.body;
    const todo = new Todo({ name });
    todo.save()
        .then(function () {
            console.log('todo item added');
            res.redirect('/todos');
        }).catch(function (e) {
            console.log(e);
            res.end();
        });
});
```

2. Showing All Todo Items

GET /todos

The following code will try to fetch all todo items and display them dynamically.

```
app.get('/todos', function  (req,res){
    res.render('index');
});
```

It is a very simple code. Now modify that code and make like on this below code so that you can retrieve all todo items in the database and pass into our ejs engine.

```
app.get('/todos', function (req, res) {
    Todo.find({})
        .then(function (todos) {
            res.render('index', { todos });
        }).catch(function (e) {
            console.log(e);
            res.end();
        });
})
```

Since we pass some data into index.ejs file, we will have to configure our HTML.

Since we can have so many to-do items to display. Hence, we need a loop inside the index.ejs file.

```
<!-- PART WHERE WE DYNAMICALLY DISPLAY DATA TO THE TABLE -->
```

Below that comment, add this code

```
  <% for(let todo of todos) { %>
        <!-- display each item into table -->
   <% } %>
```

Add the below code inside the for loop

```
<tr>
    <td>
        <p>
            <%= todo.name %>
        </p>
    </td>

    <td>
        <div class="text-center">
            <div class="mb-1">
                <form action="/todos/<%= todo._id %>?_method=DELETE"
 method="POST">
                    <button class="btn btn-danger">
                        Delete
                    </button>
                </form>
            </div>
            <div class="mb-1">
                <form action="/todos/<%= todo._id%>/edit" method="GE
T">
                    <button class="btn btn-primary">
                        Edit
                    </button>
                </form>
            </div>
        </div>
    </td>
</tr>
```

For each iteration, we put the unique link based on todo item id.

```
<div class="mb-1">
    <form action="/todos/<%= todo._id %>?_method=DELETE" method="POST">
        <button class="btn btn-danger">
            Delete
        </button>
    </form>
</div>
<div class="mb-1">
    <form action="/todos/<%= todo._id%>/edit" method="GET">
        <button class="btn btn-primary">
            Edit
        </button>
    </form>
</div>
```

Add another functionality when there is no data to be shown, show something like not todo list! Add now!

```
<!-- IF THERE IS NO TODO THEN SHOW SOMETHING -->
<% if (!todos.length> 0) { %>
    <tr>
        <td colspan="2">No todo list! Add now!</td>
    </tr>
<% } %>
```

3. Deleting Todo Item

Since each item has a unique id, we will access that to delete the item based on a unique id. We will use the delete HTTP verb. However, HTML forms only support GET and POST as HTTP request methods. To overcome it, we will use method-override. It lets you use HTTP verbs such as PUT or DELETE in places where the client doesn't support it.

To install it, type

```
Ilham@LAPTOP-0B3A56T8 MINGW64 /d/Development/express development/TodoWebApp
$ npm install method-override
```

Add the following code to our app

```
const methodOverride = require('method-override');
// override with POST having ?_method=DELETE
app.use(methodOverride('_method'))
```

If we see in index.ejs file, for delete button I have ?_method=delete to make our server think that the HTTP method is delete.

```
<div class="mb-1">
    <form action="/todos/<%= todo._id %>?_method=DELETE" method="POST">
        <button class="btn btn-danger">
            Delete
        </button>
    </form>
</div>
<div class="mb-1">
    <form action="/todos/<%= todo._id%>/edit" method="GET">
        <button class="btn btn-primary">
            Edit
        </button>
    </form>
</div>
```

In the app.js file, add the following code

Delete /todos/:id

```
// delete product
app.delete('/todos/:id', function (req, res) {
    const { id } = req.params;
    Todo.findByIdAndDelete(id)
        .then(function () {
            console.log('item deleted')
            res.redirect('/todos');
        }).catch(function (e) {
            console.log(e);
            res.end();
        });;
})
```

4. Editing Todo Item.

Firstly, we need to link the edit button to edit.ejs to show edit form as well as passing the requested todo item too.

GET /todos/:id/edit

```
// get an edit form
app.get('/todos/:id/edit', function (req, res) {
    const { id } = req.params;
    Todo.findById(id)
        .then(function (todo) {
            res.render('edit', { todo });
        }).catch(function (e) {
            res.end("<h1>oops there's an error</h1>");
        });
})
```

Make sure the form in the edit.ejs has ?_method=PUT

```
<form action="/todos/<%= todo._id%>?_method=PUT" method="POST">
    <div class="mb-3">
        <label for="name">Name:</label>
        <input class="form-control" type="text" id="name" name="name" value="<%= todo.name %> ">
    </div>

    <div class="mb-3">
        <button class="btn btn-primary">Edit</button>
    </div>
</form>
```

After that add the put request itself

PUT /todos/:id

```
//update product
app.put('/todos/:id', function (req, res) {
    const { id } = req.params;
    const { name } = req.body;
    Todo.findByIdAndUpdate(id, { name })
        .then(function () {
            res.redirect('/todos');
        }).catch(function (e) {
            console.log(e);
            res.end("<h1>oops there's an error</h1>");
        });
});
```

REFERENCES

An Absolute Beginner's Guide to Using npm. (2017). Retrieved 15 May 2021, from https://nodesource.com/blog/an-absolute-beginners-guide-to-using-npm/

Express 5.x - API Reference. (2021). Retrieved 15 May 2021, from https://expressjs.com/en/5x/api.html

HTTP Methods and why you should be using them on your API | Hacker Noon. (2021). Retrieved 15 May 2021, from https://hackernoon.com/http-methods-and-why-you-should-be-using-them-on-your-api-98e26b0a7e57

Introduction to Mongoose for MongoDB. (2018). Retrieved 15 May 2021, from https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/

The Request/Response Cycle of the Web. (2018). Retrieved 15 May 2021, from https://medium.com/@jen_strong/the-request-response-cycle-of-the-web-1b7e206e9047

Understanding HTML Form Encoding: URL Encoded and Multipart Forms. (2021). Retrieved 15 May 2021, from https://dev.to/sidthesloth92/understanding-html-form-encoding-url-encoded-and-multipart-forms-3lpa

Web Application Definition. (2014). Retrieved 15 May 2021, from https://techterms.com/definition/web_application

What is Ejs, W., & Tanoli, S. (2021). What is Ejs , What is the use of EJS?. Retrieved 15 May 2021, from https://stackoverflow.com/questions/64144316/what-is-ejs-what-is-the-use-of-ejs