Express JS

- A popular, minimalist web application framework for building APIs and web applications using Node.js.
- It simplifies the process of building web applications and APIs by providing a set of tools and utilities.

Introduction to

Express JS

Web frameworks and technologies

https://survey.stackoverflow.co/2023/#most-popular-technologies-webframe

Formål

- Formålet er at udvide vores forståelse og kompetencer indenfor backendudvikling med Node.js.
- Vi skal arbejde web frameworket Express.js, og hvad det kan bidrage med i vores Node.js backend-app.
- Praktisk tilgang: Igen fokus på opgaver kombineret med mindre teori oplæg og opsamling.

Agenda

1. Recap on Node.js
2. Introduction to Express.js
3. Basic Routing
4. Middleware, JSON Request and Response
5. Exercises

Bunden Forudsætning: Node.js & Express

FRONTEND

REST API

BACKEND

FETCH(GET, POST, PUT, DELETE)

REQUEST

JSON DATA

RESPONSE

With your own words

- What’s Node.js?
- Important features of Node.js
- What did we build in the last lecture?
- What improvements can be made?
- How can we simplify the implementation of the different concepts?

Please,

Do not try to understand everything!

Run JavaScript, not just in browsers 🎉

app.js — node-js-fun

EXPLORER

NODE-JS-FUN

app.js

app.js > ...

```
1  const yourFavoriteLecturer = {
2      name: "Peter Lind",
3      mail: "petl@kea.dk"
4  };
5
6  console.log(yourFavoriteLecturer);
7
```

PROBLEMS OUTPUT TERMINAL zsh

```
race@RACEs-MacBook-Pro node-js-fun % node app.js
{ name: 'Peter Lind', mail: 'petl@kea.dk' }
race@RACEs-MacBook-Pro node-js-fun %
```

OUTLINE

TIMELINE

0 0 Ln 7, Col 1 Spaces: 4 UTF-8 LF {} JavaScript Go Live Prettier

HTTP Server with build-in HTTP Module

app.js — hello-http-module

app.js > ...

```javascript
// Importer det indbyggede Node.js 'http' module.
import http from "http";
import fs from "fs/promises";

// Her opretter vi en HTTP-server ved at bruge 'createServer' funktionen.
// Denne funktion tager en såkaldt 'callback' funktion, der bliver kaldt, n

const app = http.createServer(async (request, response) => {
    // ROUTE: "/" - GET
    if (request.url === "/" && request.method === "GET") {
        // Sæt statuskode og overskrift for responsen
        response.statusCode = 200;
        response.setHeader("Content-Type", "text/plain");
        // Send besked som response
        response.end("Working with HTTP Module and routing");
    }
    // ROUTE: "/users" - GET
    else if (request.url === "/users" && request.method === "GET") {
        // Sæt statuskode og overskrift for responsen
        response.statusCode = 200;
        response.setHeader("Content-Type", "application/json");
        const json = await fs.readFile("data/users.json");
        // Send JSON som response
        response.end(json);
    }
    // ROUTE: "/posts" - GET
    else if (request.url === "/posts" && request.method === "GET") {
        // Sæt statuskode og overskrift for responsen
        response.statusCode = 200;
        response.setHeader("Content-Type", "application/json");
        const json = await fs.readFile("data/posts.json");
        // Send JSON som response
        response.end(json);
    }

    // ROUTE: "/users" - POST
    else if (request.url === "/users" && request.method === "POST") {
```

file-system 0 0 Ln 6, Col 87 Spaces: 4 UTF-8 LF JavaScript Go Live Prettier

Routing with HTTP Module

app.js — node-vanilla-todos-rest-api

app.js > app > http.createServer() callback

```javascript
import http from "http";
import fs from "fs/promises";

const app = http.createServer(async (req, res) => {
    // "/"" : GET
    if (req.url === "/" && req.method === "GET") {…
    }
    // "/todos" : GET
    else if (req.url === "/todos" && req.method === "GET") {…
    }

    // "/todos" : POST
    else if (req.url === "/todos" && req.method === "POST") {…
    }

    // "/todos/:id" : PUT
    else if (req.url.match(/\/todos\/([0-9]+)/) && req.method === "PUT") {…
    }

    // "/todos/:id" : DELETE
    else if (req.url.match(/\/todos\/([0-9]+)/) && req.method === "DELETE")
    }

    // If no route present
    else {…
    }
});

app.listen(3000, () => {
    console.log(`App running on http://localhost:3000`);
});
```

main 0 0 Ln 6, Col 3 Spaces: 4 UTF-8 LF JavaScript Go Live Prettier

- Request and response headers, Content-Type
- Status Codes

app.js — hello-http-module

app.js > app > http.createServer() callback

```javascript
// Importer det indbyggede Node.js 'http' module.
import http from "http";
import fs from "fs/promises";

// Her opretter vi en HTTP-server ved at bruge 'createServer' funktionen.
// Denne funktion tager en såkaldt 'callback' funktion, der bliver kaldt

const app = http.createServer(async (request, response) => {
    // ROUTE: "/" - GET
    if (request.url === "/" && request.method === "GET") {
        // Sæt statuskode og overskrift for responsen
        response.statusCode = 200;
        response.setHeader("Content-Type", "text/plain");
        // Send besked som response
        response.end("Working with HTTP Module and routing");
    }
    // ROUTE: "/users" - GET
    else if (request.url === "/users" && request.method === "GET") {
        // Sæt statuskode og overskrift for responsen
        response.statusCode = 200;
        response.setHeader("Content-Type", "application/json");
        const json = await fs.readFile("data/users.json");
        // Send JSON som response
        response.end(json);
    }
    // ROUTE: "/posts" - GET
    else if (request.url === "/posts" && request.method === "GET") {
        // Sæt statuskode og overskrift for responsen
        response.statusCode = 200;
        response.setHeader("Content-Type", "application/json");
        const json = await fs.readFile("data/posts.json");
        // Send JSON som response
        response.end(json);
    }
```

file-system 0 0 Ln 18, Col 69 Spaces: 4 UTF-8 LF JavaScript Go Live Prettier

Getting Request data from Request body

app.js — node-vanilla-todos-rest-api

app.js > app > http.createServer() callback > data

```javascript
    }
    // "/todos" : POST
    else if (req.url === "/todos" && req.method === "POST") {
        // get the data sent along
        const newTodo = JSON.parse(await getReqData(req));
        // todo object with generated dummy id and parsed todoData
        const todo = {
            id: new Date().getTime(),
            ...newTodo
        };
        const data = await fs.readFile("data.json");
        const todos = JSON.parse(data);
        todos.push(todo);
        const todosJSON = JSON.stringify(todos);
        await fs.writeFile("data.json", todosJSON);

        // set the status code and content-type
        res.writeHead(200, { "Content-Type": "application/json" });
        //send the todo
        res.end(todosJSON);
    }
});

app.listen(3000, () => {
    console.log(`App running on http://localhost:3000`);
});

async function getReqData(req) {
    return new Promise((resolve, reject) => {
        let body = "";
        req.on("data", chunk => (body += chunk));
        req.on("end", () => resolve(body));
        req.on("error", reject);
```

... we can use our existing **JavaScript** **expertise** to develop backend applications, servers, REST APIs, etc.

Runtime
Environment

Express JS

- A popular, minimalist web application framework for building APIs and web applications using Node.js.
- It simplifies the process of building web applications and APIs by providing a set of tools and utilities.

Express JS

A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications

Express JS

Express.js is known for its flexibility and ease of use, making it an ideal choice for developers who want to build web applications quickly and efficiently. With its built-in middleware functions, routing system, and template engines, Express.js simplifies the process of building complex web applications.

My Project

NODE-USERS-REST-API

- node_modules
- .gitignore
- app.js
- data.json
- package-lock.json
- package.json

npm

Express JS

CORS

And many other
JavaScript libraries

- **Single-page applications (SPAs)**
  Express.js creates SPAs that load all resources on a single page, ensuring a seamless user experience.
- **Mobile applications**
  Express.js builds mobile apps using web technologies like HTML, CSS, and JavaScript.
- **RESTful APIs**
  Express.js crafts RESTful APIs for data manipulation and access by other applications.
- **Server-side rendering**
  Express.js enables server-side rendering, rendering pages on the server before sending to the client.
- **Real-time applications**
  Express.js constructs real-time apps, facilitating ongoing client-server communication.
- **Microservices**
  Express.js constructs microservices, small independent units combined to create larger applications.

Express JS

What is it used for?

- **Minimalistic Framework**
  Express is a lightweight and minimal web application framework for Node.js.
- **Routing**
  It offers a powerful routing system to handle different HTTP methods and URL patterns.
- **Middleware**
  Express uses middleware to handle various tasks, such as authentication, logging, and data parsing.
- **HTTP Utility Methods**
  Provides easy-to-use methods for interacting with the HTTP protocol.
- **Extensibility**
  Allows integration of third-party middleware and extensions for added functionality.
- **Static File Serving**
  Simplifies serving static files like CSS, JavaScript, and images.
- **Error Handling**
  Offers error handling mechanisms for a better user experience.
- **Flexible for APIs**
  Suitable for building RESTful APIs with its routing and middleware capabilities.
- **Easy to Use**
  Express.js is easy to use and requires minimal configuration, making it ideal for developers who want to build web applications quickly.

Express JS

Key Features

Streamlines and simplifies the implementation of REST API

Express JS

Routing

What’s a Router?

What’s a Router?

“It is the piece of software in charge to organize the states of the application, switching between different views.”

Routes organize how your application handles different requests, making your code structured and maintainable.

https://medium.com/@fro_g/routing-in-javascript-d552ff4d2921

/create
route
Router
(router.js)
view
view
view
users
page
create
page
update
page

Server

Client

Incoming request URL

Router

GET “/posts”

POST “/posts”

PUT “/posts”

DELETE “/posts”

Based on the request URL and method, the router decides what to do

```
app.get("/posts", (req, res) => {
    // return all posts
});

app.post("/posts", (req, res) => {
    const post = req.body;
    // Create a new post based on th
});

app.put("/posts/:id", (req, res) =>
    const postId = req.params.id;
    const post = req.body;
    // Find and update the post with
});

app.delete("/posts/:id", (req, res)
    const postId = req.params.id;
    // Find and delete the post with
});
```

Routes in Express.js

In Node.js with Express.js, routes are a fundamental concept that helps you define **how your web application responds to different types of HTTP requests** at different URLs. Think of routes as the paths your application can take based on the URLs that users request.

Routes in Express.js

- The Express routing is built upon Node.js HTTP Module and it **simplifies the process of handling different URL paths and HTTP methods** in your web application.
- Express routing enables you to define routes for different endpoints, **simplifying request handling**.
- This enhances and streamlines code organization, and readability, and lets you focus on building functionality rather than intricate routing logic.

Routes in Express.js

- Routes determine how your application responds to specific URLs and HTTP methods.
- Each route combines an HTTP method (like GET, POST) with a URL path.
- When a client request matches a route, a corresponding route handler function is executed to process the request and send a response.

```javascript
import express from "express";
const app = express();

// Define routes using app.METHOD(PATH, HANDLER)
app.get("/posts", (req, res) => {
    // return all posts
});

app.post("/posts", (req, res) => {
    const post = req.body;
    // Create a new post based on the data in req.body
});

app.put("/posts/:id", (req, res) => {
    const postId = req.params.id;
    const post = req.body;
    // Find and update the post with postId and data from req.body
});

app.delete("/posts/:id", (req, res) => {
    const postId = req.params.id;
    // Find and delete the post with postId
});

// ...and so on

app.listen(3000, () => {
    console.log("Server is running on port 3000");
});
```

Express JS

Middleware, JSON Request and Response

Web Development

WEB APP

REST API

DATA SOURCE

FETCH(GET, POST, PUT, DELETE)

REQUEST {JSON DATA}

RESPONSE {JSON DATA}

Express

CLIENT

SERVER

- In Express.js, middleware refers to functions that are **executed during the request-response cycle** of a web application. These functions have access to the request and response objects and can modify them, perform tasks, or control the flow of the application.
- Express.js allows you to add middleware at the application level using `app.use()` or at specific route levels. Middleware functions are executed in the order they are added, so the order matters.
- For instance, using `app.use(express.json())` is a middleware that automatically parses incoming JSON data from requests, making it accessible as JavaScript objects in your route handlers. This simplifies handling JSON data.
- Similarly, `app.use(cors())` is middleware that enables Cross-Origin Resource Sharing (CORS), allowing your server to respond to requests from different domains. This is essential for security when your frontend and backend are on separate domains.
- In both cases, middleware helps simplify and enhance the functionality of your Express application by handling common tasks automatically before your route handlers are executed.

app.js — node-users-rest-api

app.js > ...

```javascript
app.use(express.json()); // to parse JSON bodies
app.use(cors()); // Enable CORS for all routes

app.listen(port, () => {
    console.log(`App running on http://localhost:${port}`);
});

app.get("/", (request, response) => {
    response.send("Node.js Users REST API 🎉");
});

// READ all users - "/users" : GET
app.get("/users", async (request, response) => {
    const data = await fs.readFile("data.json");
    const users = JSON.parse(data);
    response.json(users);
});

// CREATE user - "/users" : POST
app.post("/users", async (request, response) => {
    const newUser = request.body;
    newUser.id = new Date().getTime();
    console.log(newUser);

    const data = await fs.readFile("data.json");
    const users = JSON.parse(data);

    users.push(newUser);
    console.log(newUser);
    fs.writeFile("data.json", JSON.stringify(users));
    response.json(users);
});
```

main+ ⊗ 0 ⚠ 0 Spaces: 4 UTF-8 LF {} JavaScript Go Live Prettier

node JS

HTTP Module

—>

node JS

+

express

Simple GET Request

```
const app = http.createServer(async (request, response) => {
    // READ all users - "/users" : GET
    if (request.url === "/users" && request.method === "GET") {
        const data = await fs.readFile("data.json");
        response.writeHead(200, { "Content-Type": "application/json" });
        response.end(data);
    }
```

—>

```
// READ all users - "/users" : GET
app.get("/users", async (request, response) => {
    const data = await fs.readFile("data.json");
    const users = JSON.parse(data);
    response.json(users);
});
```

“Simple” POST Request

```javascript
const app = http.createServer(async (request, response) => {
    // "/users" : POST
    if (request.url === "/users" && request.method === "POST") {
        // get the data sent along
        const newUser = JSON.parse(await getReqData(request));
        // user object with generated dummy id and parsed userData
        const user = {
            id: new Date().getTime(),
            ...newUser
        };
        const data = await fs.readFile("data.json");
        const users = JSON.parse(data);
        users.push(user);
        const usersJSON = JSON.stringify(users);
        await fs.writeFile("data.json", usersJSON);

        // set the status code and content-type
        response.writeHead(200, { "Content-Type": "application/json" });
        //send the user
        response.end(usersJSON);
    }
});

async function getReqData(req) {
    return new Promise((resolve, reject) => {
        let body = "";
        req.on("data", chunk => (body += chunk));
        req.on("end", () => resolve(body));
        req.on("error", reject);
    });
}
```

—>

```javascript
// CREATE user - "/users" : POST
app.post("/users", async (request, response) => {
    const newUser = request.body;
    newUser.id = new Date().getTime();

    const data = await fs.readFile("data.json");
    const users = JSON.parse(data);

    users.push(newUser);
    fs.writeFile("data.json", JSON.stringify(users));
    response.json(users);
});
```

Exercises

- Getting Started with Express.js
- From Node.js HTTP Module to Express.js

POSTMAN

Getting Started with Postman

- Postman is a software tool for working with APIs.
- We can use it as client to test our backend API’s and to test APIs in general.
- It helps developers send requests and receive responses from APIs.
- Postman tests API functionality and documents how to use APIs.
- It simplifies API development and testing.
- Developers collaborate using Postman to work together effectively.

Web Development

POSTMAN

FIREBASE REST API

DATA SOURCE

CLIENT

SERVER

Client

REQUEST

RESPONSE

Backend

Code Every Day