HyperionDev

Full-Stack Web Development

HyperionDev

Building RESTful APIs with Express.js

Full Stack Web Development Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Full Stack Web Development Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support
- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting
- We would love your **feedback** on lectures: Feedback on Lectures

Objectives

- ❖ Set up a basic Express.js server.
- ❖ Implement server-side routing.
- ❖ Build and test RESTful APIs using Express.js.

WHAT ARE RESTFUL APIS WITH EXPRESS.JS?

RESTful APIs with Express.js refer to building and maintaining server-side endpoints that adhere to the REST architectural principles.

They involve creating routes that handle HTTP requests (like GET, POST, PUT, DELETE) to interact with data stored on the server.

This approach is essential for enabling communication between a client (like a web or mobile app) and a server, allowing the client to perform operations such as retrieving, creating, updating, or deleting data.

Express.js, a Node.js framework, simplifies the process of setting up and managing these RESTful endpoints, making it a popular choice for building efficient and scalable web services.

Key aspects of RESTful APIs

- ❖ Aspects RESTful APIs
  - ➢ Routing: (Setting up URL paths to handle HTTP requests like GET,POST,PUT and DELETE)
  - ➢ Middleware: (Use functions for tasks like, logging, authentication, and data parsing.
  - ➢ Data Handling: Manage CRUD operations on the server side data
  - ➢ REST Principles: Follow guidelines for scalable and consistent APIs
  - ➢ Error Handling: Provide meaningful feedback for clients requests
  - ➢ Security: implement Authentication
- ❖ RESTful APIs with Express.js enable efficient communication between client and server, ensuring smooth data exchange and secure operations in web applications.

REST Principles Overview

- **REST** principles guide the design of APIs to ensure they are scalable stateless and resource-oriented

- **Key Principles:**
  - Statelessness
  - Client server architecture
  - Uniform Interface
  - Resource based URLs
  - HTTP Methods
  - Layered System

Statelessness

- **Concept:** Each request from a client to a server must contain all the information the server needs to fulfill the request

- **Benefits:**
  - Scalability No need to store session information, making the system easier to scale
  - Simplicity: simplifies server logic

Client-Server Architecture

- **Concept:** The client and server operate independently, with the client handling the user interface (UI) and the server managing data and business logic

- **Benefits:**
  - Modularity: Client and server can evolve separately
  - Interoperability: Clients from different platforms can interact with the server.

Uniform Interface

- ❖ **Concept:** A standardised way of interacting with resources via defined URIs and HTTP methods.

- ❖ **Key constraints:**
  - ➢ Resource Identification: Use URIs to identify resources
  - ➢ Self-descriptive Messages: Requests and responses contain enough information to describe how to process them.
  - ➢ Documentation: links in responses guide the client on how to use the API

- ❖ **Benefits:**
  - ➢ Predictability: easier to interact with the API
  - ➢ Decoupling: The client and server remain loosely coupled

Resource-Based URLs

- ❖ **Concept:**
  - ➢ Resources are identified and accessed via URIs
- ❖ Example:
  - ➢ `/users/1234`: Access a specific user
  - ➢ `/products/`: Retrieve a list of products
- ❖ **Benefits:**
  - ➢ Clarity: Clear and Meaningful URLs
  - ➢ Scalability: Independent management of resources

HTTP METHODS

- **Concept:**
  - Use standard HTTP methods to perform operations on resources
- **Methods:**
  - GET: Retrieve a resource
  - POST: Create a new resource
  - PUT: Update an existing resource
  - DELETE: Remove a resource
  -
- **Benefits:**
  - Predictability: Consistent behaviour across APIs
  - Interoperability: HTTP is universally supported

Layered Systems

- ❖ **Concept:**
  - ➢ RESTful systems can be composed of multiple layers (client, proxy, server), with each layer only interacting with its adjacent layers.

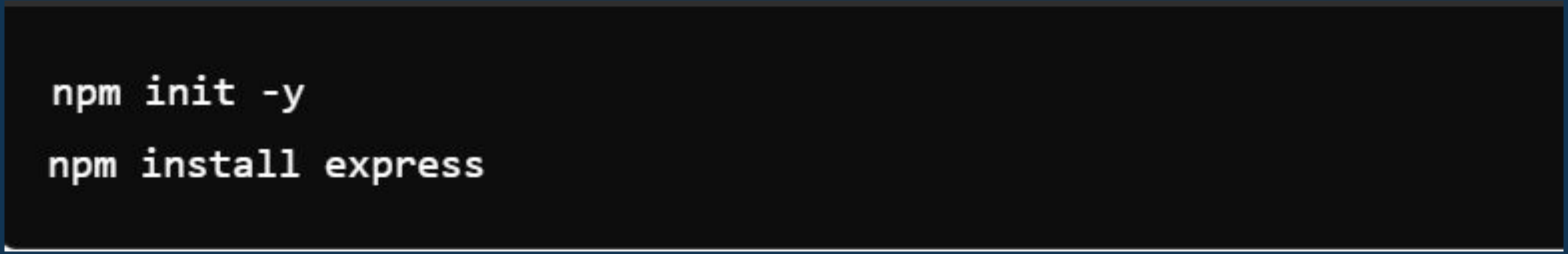
- ❖ **Benefits:**
  - ➢ **Scalability**: Each layer can be scaled independently.
  - ➢ **Security**: Layers can enforce security measures.

Q&A

- Please ask any questions if you want any clarification

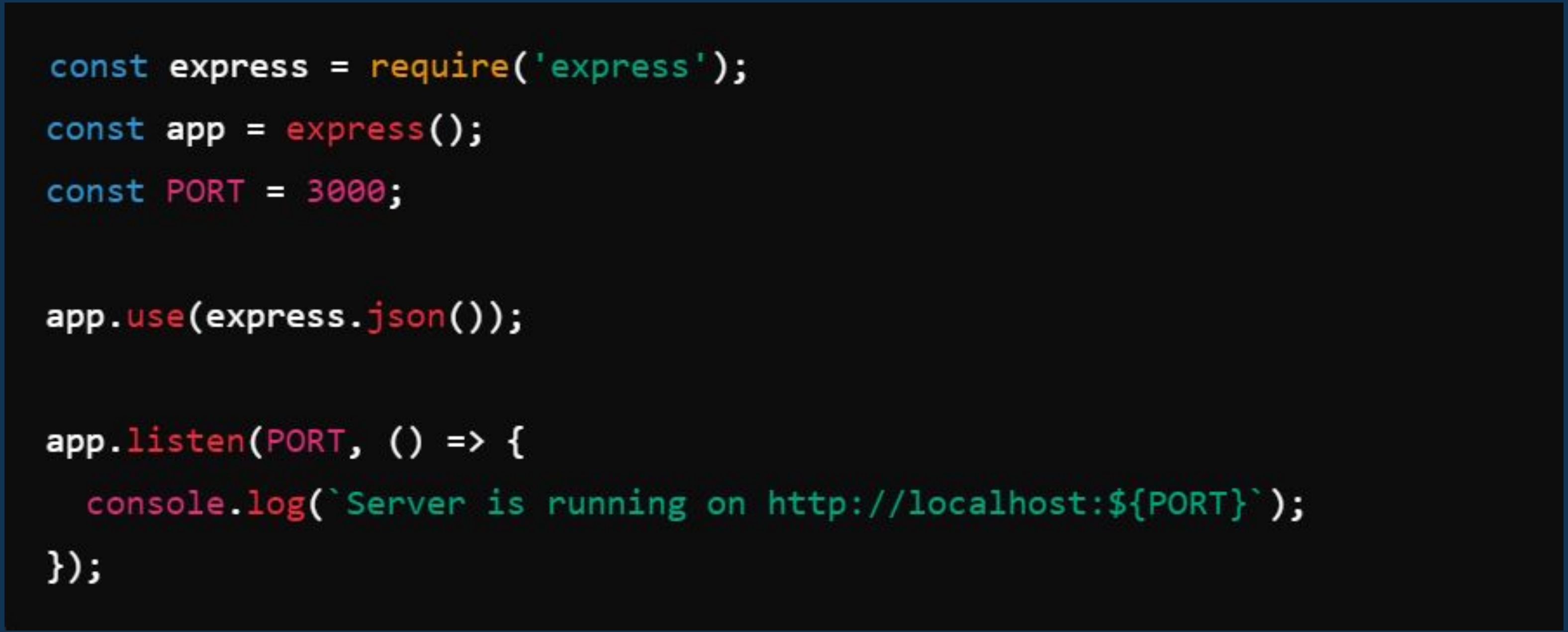
Creating a simple CRUD Application

- ❖ Setup (10 mins)
- ❖ Install Express



```
npm init -y
npm install express
```

SETUP OF BASIC SERVER



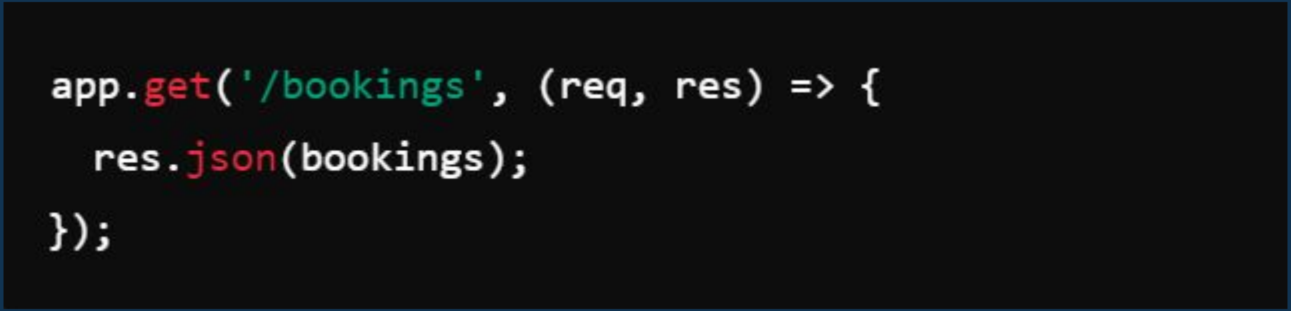
```javascript
const express = require('express');
const app = express();
const PORT = 3000;

app.use(express.json());

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

Implement Bookings Endpoints

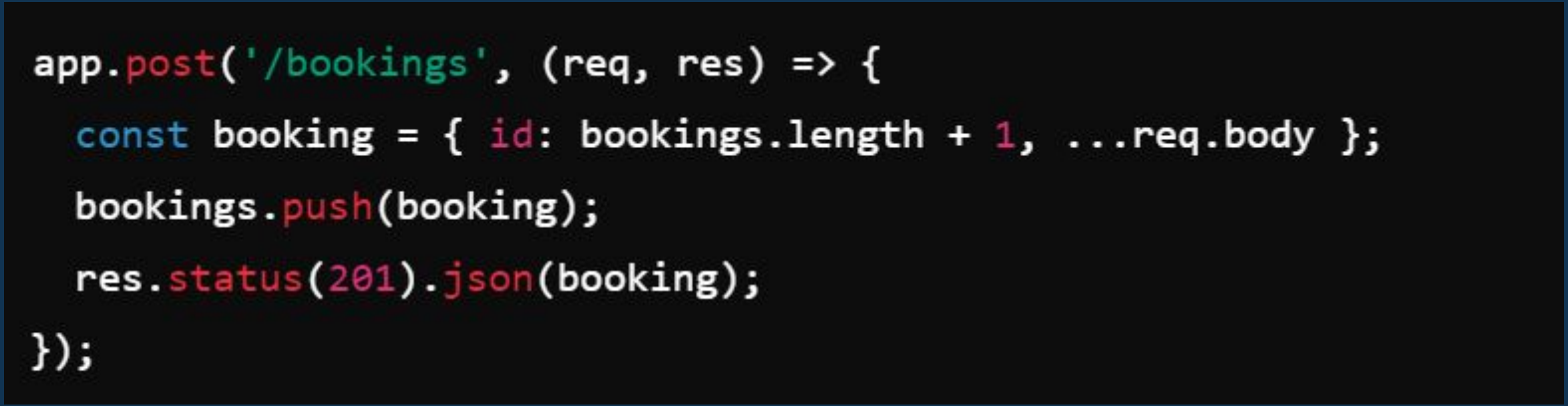
GET/Bookings



```
app.get('/bookings', (req, res) => {
  res.json(bookings);
});
```

Implement Bookings Endpoints

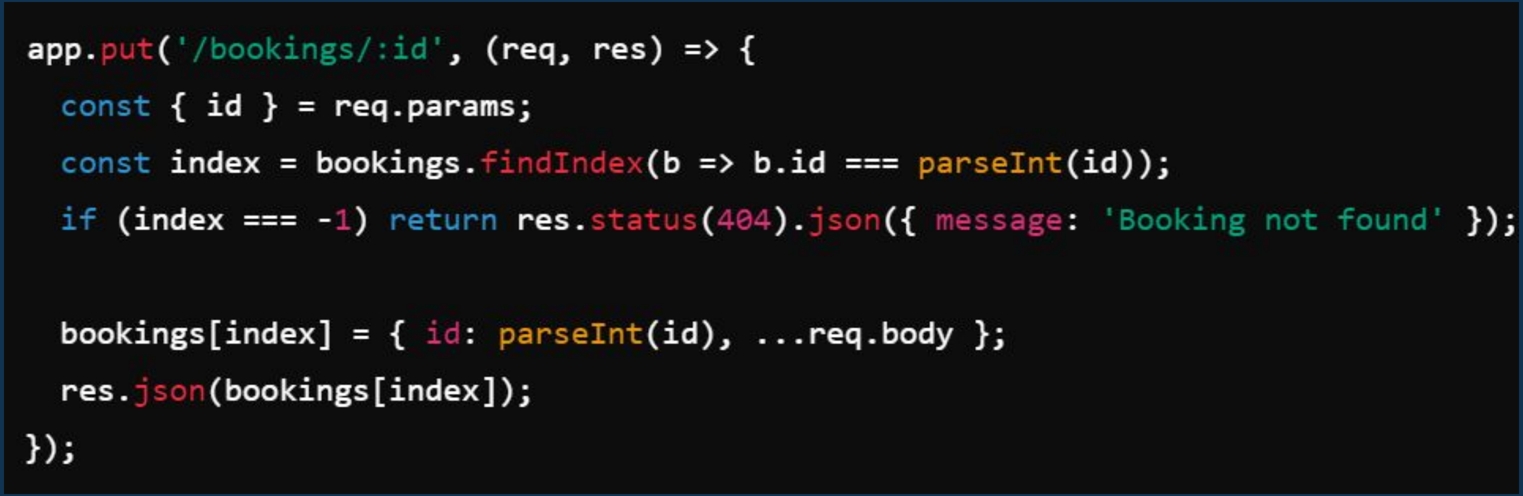
POST/bookings



```
app.post('/bookings', (req, res) => {
  const booking = { id: bookings.length + 1, ...req.body };
  bookings.push(booking);
  res.status(201).json(booking);
});
```

Implement Bookings Endpoints

PUT/bookings

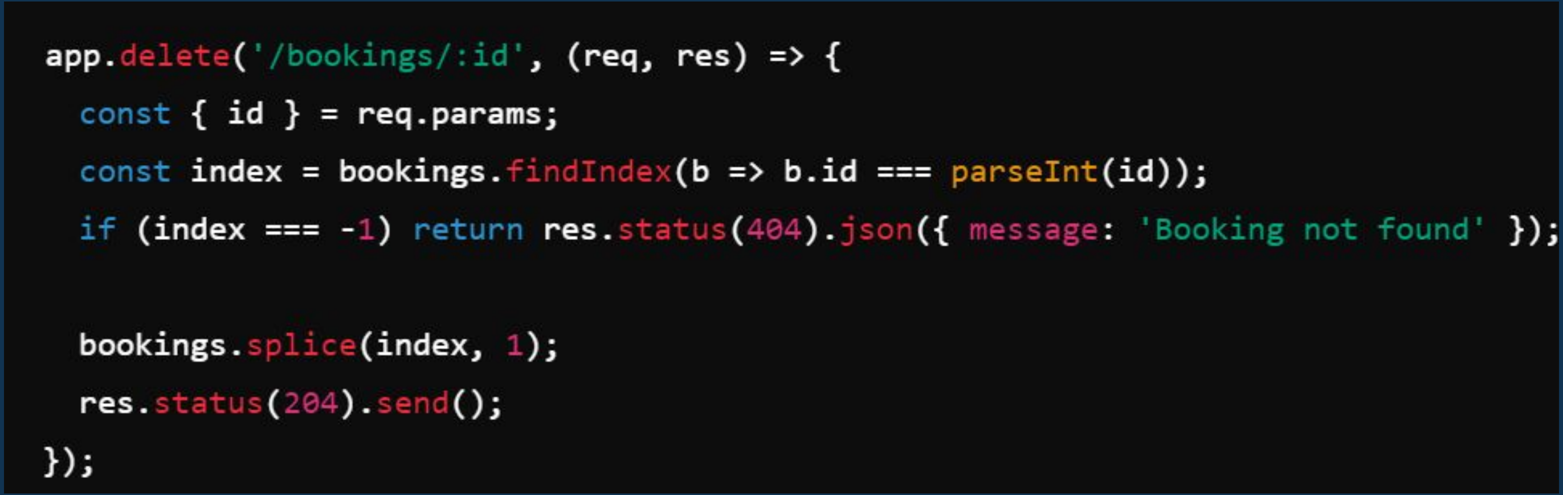


```
app.put('/bookings/:id', (req, res) => {
  const { id } = req.params;
  const index = bookings.findIndex(b => b.id === parseInt(id));
  if (index === -1) return res.status(404).json({ message: 'Booking not found' });

  bookings[index] = { id: parseInt(id), ...req.body };
  res.json(bookings[index]);
});
```

Implement Bookings Endpoints

DELETE/bookings



```
app.delete('/bookings/:id', (req, res) => {
  const { id } = req.params;
  const index = bookings.findIndex(b => b.id === parseInt(id));
  if (index === -1) return res.status(404).json({ message: 'Booking not found' });

  bookings.splice(index, 1);
  res.status(204).send();
});
```

Using Postman to Test RESTful APIs

- **What is Postman?**
  - Postman is a powerful tool for API development and testing.
  - It allows users to send requests to APIs and view responses in a user-friendly interface.
- **Key Features:**
  - **User Interface**: Intuitive interface for constructing requests and viewing responses.
  - **HTTP Methods**: Supports all HTTP methods (GET, POST, PUT, DELETE, etc.).
  - **Environment Variables**: Allows storing variables for reuse in different requests.
  - **Collections**: Organize requests into groups for better management.

Using Postman to Test RESTful APIs

Steps to Use Postman:

1. **Install Postman**: Download and install Postman from the official website.
2. **Create a New Request**:
   - Click on "New" and select "HTTP Request".
3. **Select the HTTP Method**:
   - Choose the method you want to use (GET, POST, etc.) from the dropdown.
4. **Enter the Request URL**:
   - Input the URL of the API endpoint you want to test.
5. **Add Request Body** (for POST/PUT requests):
   - Go to the "Body" tab and select "raw" or "form-data".
   - Enter the required data in JSON format if using "raw".
6. **Send the Request**:
   - Click the "Send" button to send the request to the server.
7. **View the Response**:
   - Check the "Response" section for status code, response time, and data returned from the server.

Using Postman to Test RESTful APIs

- **Tips for Using Postman:**
  - Use **Collections** to group related API requests for easy access.
  - Utilize **Environment Variables** for dynamic data (e.g., tokens, URLs).
  - Explore **Test Scripts** to automate testing of responses.

Hyperiondev

Questions and Answers

HyperionDev

Thank You for attending!