




Welcome to the CoGrammar

Lecture: Database Interaction and the MERN stack

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated
moderators answering questions.



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](#)

Skills Bootcamp

8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✓ Criterion 1: Initial Requirements

Timeframe: First 2 Weeks

Guided Learning Hours (GLH):

Minimum of 15 hours

Task Completion: First four tasks

Due Date: 24 March 2024

✓ Criterion 2: Mid-Course Progress

60 Guided Learning Hours

Data Science - **13 tasks**

Software Engineering - **13 tasks**

Web Development - **13 tasks**

Due Date: 28 April 2024

Skills Bootcamp Progression Overview

✓ Criterion 3: Course Progress

Completion: All mandatory tasks,
including Build Your Brand and
resubmissions by study period end
Interview Invitation: Within 4 weeks
post-course
Guided Learning Hours: Minimum of
112 hours by support end date
(10.5 hours average, each week)

✓ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship
Outcome: Document within 12
weeks post-graduation
Relevance: Progression to
employment or related
opportunity

Lesson Objectives

- ❖ Connect with the MongoDB database using Mongoose
- ❖ Creating schemas and models using mongoose
- ❖ Designing RESTful APIs and perform CRUD functionalities with Express.js and Mongoose.
- ❖ Testing RESTful APIs with Postman.
- ❖ Understand the MERN stack architecture.

**SKILLS
FOR LIFE**

SKILLS BOOTCAMPS



Department
for Education

CoGrammar

Database Interaction(Mongoose) and the MERN stack

May 2024

Introduction to Mongoose



Database Interaction


Introduction to Mongoose

- ❖ Interacting directly with MongoDB in Node.js can be cumbersome and tedious due to its **raw API** and **lack of data validation**.
- ❖ Mongoose comes into play and provides a **higher-level of abstraction** for MongoDB by **simplifying database interactions** and enhancing developer productivity.
- ❖ Mongoose is an **object data model (ODM)**, tool that allows the programmer to treat documents stored in databases as JavaScript objects.



Database Interaction

Key Features of Mongoose

- ❖ **Data validation:** This ensures that data adheres to specific criteria before being saved to the database.
 - ❖ **Schema Definition:** This allows developers to define schemas that represent the structure of data stores in MongoDB collections.
 - ❖ **Query building:** Provides rich API for building MongoDB queries like filtering, sorting, updating and deleting.
 - ❖ **Model creation:** Mongoose offer creation of models that serve as constructors for Mongodb documents.
- 

Mongoose

Installation

- ❖ Prerequisites:
 - Have a Node.js server with express installed.
 - Have MongoDB server configured locally.
 - Have a MongoDB atlas account with a cluster running.
- ❖ Command for installing mongoose
 - `npm install mongoose`

Mongoose

Connecting to database

- ❖ After installing Mongoose, you need to connect to a MongoDB database. We'll import the Mongoose module and utilise it.

```
index.js

1  const mongoose = require("mongoose");
2  const URL =
3    "mongodb+srv://{username}:{password}@cluster0.vrj2mpt.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0";
4  const clientOptions = {
5    serverApi: { version: "1", strict: true, deprecationErrors: true },
6  };
7
8  mongoose
9    .connect(URL, clientOptions)
10   .then(() => {
11     console.log("Connected successfully...");
12   })
13   .catch((err) => {
14     console.log("Error connecting to db", err);
15   });
16
```

Mongoose

Connecting to database

- ❖ The connect method takes in two arguments, the **URI connection string** and **client options**.
- ❖ The chaining method using the **.then** and **.catch** methods are set for error handling incase the connection is not made successfully.

Creating Schemas

Using mongoose to create schemas

- ❖ A **schema** in MongoDB represents the **structure** and **organization** of data within a collection.
- ❖ The **mongoose.Schema()** method takes in an object as an argument that contains the **name of data** to be stored as the key and its **datatype** as the value in the object.

Creating Schemas

Using mongoose to create schemas


schema.js

```
1  const mongoose = require('mongoose')
2
3  //creates the collection structure of a blog
4  const blogSchema = mongoose.Schema({
5    //the title of a blog (a string and is required)
6    title: {
7      required : true,
8      type: String,
9
10   },
11   //the blog content (a string and is required)
12   content: {
13     type: String,
14     required: true
15   },
16   //the blog author (a string and is required)
17   author: {
18     type: String,
19     required: true
20   },
21   //the blog creation time (a date field, auto created itself.)
22   createdAt: {
23     type: Date,
24     required: true,
25     default: Date.now
26   }
27 })
```



Creating a model

Using mongoose to create schemas

- ❖ Models in MongoDB represent documents which can be saved and retrieved from the database.
 - ❖ We create models using the **mongoose.model()** method, it takes in two arguments:
 - **Model name:** This is a string that represents the document in our database and how we'll refer to it.
 - **The schema object:** The structure of the document will be defined by the schema object we just created.
- 

Creating Schemas

Using mongoose to create schemas

schema.js

```
29 const BlogModel = mongoose.model('Blog', blogSchema)
```

Performing CRUD operations with Mongoose.



Prerequisites

Setup needed for CRUD functionality

- ❖ We've created the following so far:
 - **A Blog Model/document:** To store blogs in our database.
 - **A blog schema:** The structure of the blog document
- ❖ In order to interact with the database via API endpoints, we also need to set up express to have routes.
 - Install Express.js and configure a server
 - Configure routes to access the CRUD endpoints.
- ❖ The next slide is a demonstration of the full configuration of Mongoose and Express.

Configuration

Setting up Express.js and MongoDB

```
index.js

1  const mongoose = require("mongoose"); //mongoose import
2  const express = require("express"); //express import
3
4  //express configuration
5  const app = express();
6  app.use(express.json());
7
8  app.listen(8000, () => {
9    console.log("Server is running on port http://localhost:8000");
10 });
11
12 //mongoose configuration
13 const URL =
14   "mongodb+srv://{username}:{password}@cluster0.vrj2mpt.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0";
15 const clientOptions = {
16   serverApi: { version: "1", strict: true, deprecationErrors: true },
17 };
18
19 mongoose
20   .connect(URL, clientOptions)
21   .then(() => {
22     console.log("Connected successfully...");
23   })
24   .catch((err) => {
25     console.log("Error connecting to db", err);
26   });
```


CREATE

```
●●● schema.js

29 const BlogModel = mongoose.model('Blog', blogSchema)
30
31 const createBlog = async (req, res)=>{
32   try {
33     const { title, content, author } = req.body
34     const blog = await BlogModel.create({
35       title: title,
36       content: content,
37       author: author
38     })
39     const savedBlog = await blog.save()
40     console.log(savedBlog)
41     res.send('Blog created successfully')
42
43   } catch(error) {
44     console.log(error)
45   }
46 }
47
48 app.post("/create", createBlog);
```

READ

```
●●● schema.js

48 //getting all blogs
49 const getAllBlogs = async (req, res)=>{
50   try {
51     const blogs = await BlogModel.find()
52     res.json(blogs)
53
54
55   } catch(error) {
56     console.log(error)
57   }
58 }
59
60 app.get('/blogs', getAllBlogs)
```

UPDATE

schema.js

```
60 //update a specific blog
61 const updateBlog = async (req, res)=>{
62   const { id } = req.params
63   const { title, content, author } = req.body
64   try {
65     const blog = await BlogModel.findByIdAndUpdate(id, {title, content, author}, { new: true})
66     res.json(blog)
67   } catch (error) {
68     console.log(error)
69   }
70 }
71
72 app.put('/update/:id', updateBlog)
```

DELETE

schema.js

```
72 //deletes a specific blog
73 const deleteBlog = async (req, res)=>{
74     const { id } = req.params
75     try {
76         const blog = await BlogModel.findByIdAndDelete(id)
77         res.json(blog)
78     } catch (error) {
79         console.log(error)
80     }
81 }
82 }
83
84 app.delete('/delete/:id', deleteBlog)
```

The MERN stack



MERN

The MERN stack

- ❖ The acronym stands for
 - **M**ongoDB: The database
 - **E**xpress.js: nodejs framework for providing robust server side functionalities and logic.
 - **R**ect: Javascript client side library, builds our frontend
 - **N**ode.js: a JavaScript runtime environment.

MERN

The MERN stack (Backend Structure)

- ❖ While combining the tools, since the structure of the Express framework is unopinionated, you are responsible for structuring your own code in any desired format.
- ❖ Although you have control, it is advisable to keep your structure based on a recognised software architecture.
- ❖ Most backend frameworks use the MVC architecture.

MVC

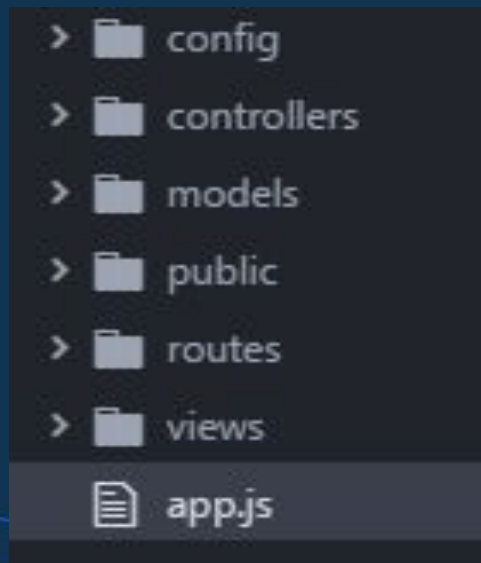
MVC architecture

- ❖ **Models:** Handles data access
- ❖ **Views:** Renders the user interface based on data received from the controller
- ❖ **Controllers:** Process user requests, interacts with the Model and View, and orchestrates the flow of data within the application.

MVC

MVC architecture

- ❖ A project with the MVC architecture will have a similar structure as this.



Questions and Answers



Thank you for attending



Department
for Education

CoGrammar

