Introduction To Mongoose

Software Development Bootcamp

Using Mongoose With MongoDB and Express

What Is Mongoose?

- Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js
- It provides a higher-level abstraction for interacting with MongoDB, including:
  - Schema definitions
  - Data validation
  - Query building

Why Use Mongoose?

Mongoose replaces the mongodb driver and:

- Executes mongodb queries on your behalf
- Provides schema-based data modeling
- Ensures data integrity

Topic

Schemas And Models

What Are Mongoose Schemas?

- A schema defines the structure of the data in a collection and can also define methods.
- Schemas:
  - Define data structure and behavior
  - Create a model
  - Use SchemaTypes

Upright Education

Example Schema

For more details see the Mongoose documentation

```
import { Schema, model } from "mongoose";

const studentSchema = new Schema({
   firstName: {
       type: String,
       required: true
   },
   lastName: {
       type: String,
       required: true
   },
   email: {
       type: String,
       required: true,
       unique: true
   },
   password: {
       type: String,
       required: true
   }
})

export default model("Student", studentSchema)
```

Why Use Schemas?

- Ensures data integrity by enforcing a consistent structure
- Prevents invalid data from being saved to the database
- Reduces the need for manual data checking in your application code
- Provides clear error messages when validation fails
- Allows for custom validation rules to meet specific business requirements
- Improves overall data quality and reliability in your application

What Are Mongoose Models?

- A Model is an object created using the schema
- Models:
    - Enforce schema types (e.g., String, Boolean)
    - Create a collection based on the model name
    - Instances of models are documents

Models And Collections

- Creating a model instance makes a document
- Using `.save()` adds the document to a collection
- Collection names are lowercase and pluralized versions of the model name (e.g. **Student** becomes **students**)

Topic

Using Mongoose And MongoDB

General Setup

1. **Initial setup**
    a. Create new directory
    b. install dependencies
2. **Server Setup**
    a. Make necessary changes to **package.json**
    b. Set up **.gitignore** and **.env** files
    c. Import necessary tools (**.env, express, mongoose, cors**)
    d. Connect Database / Add Middleware
3. **Route And Model Setup**

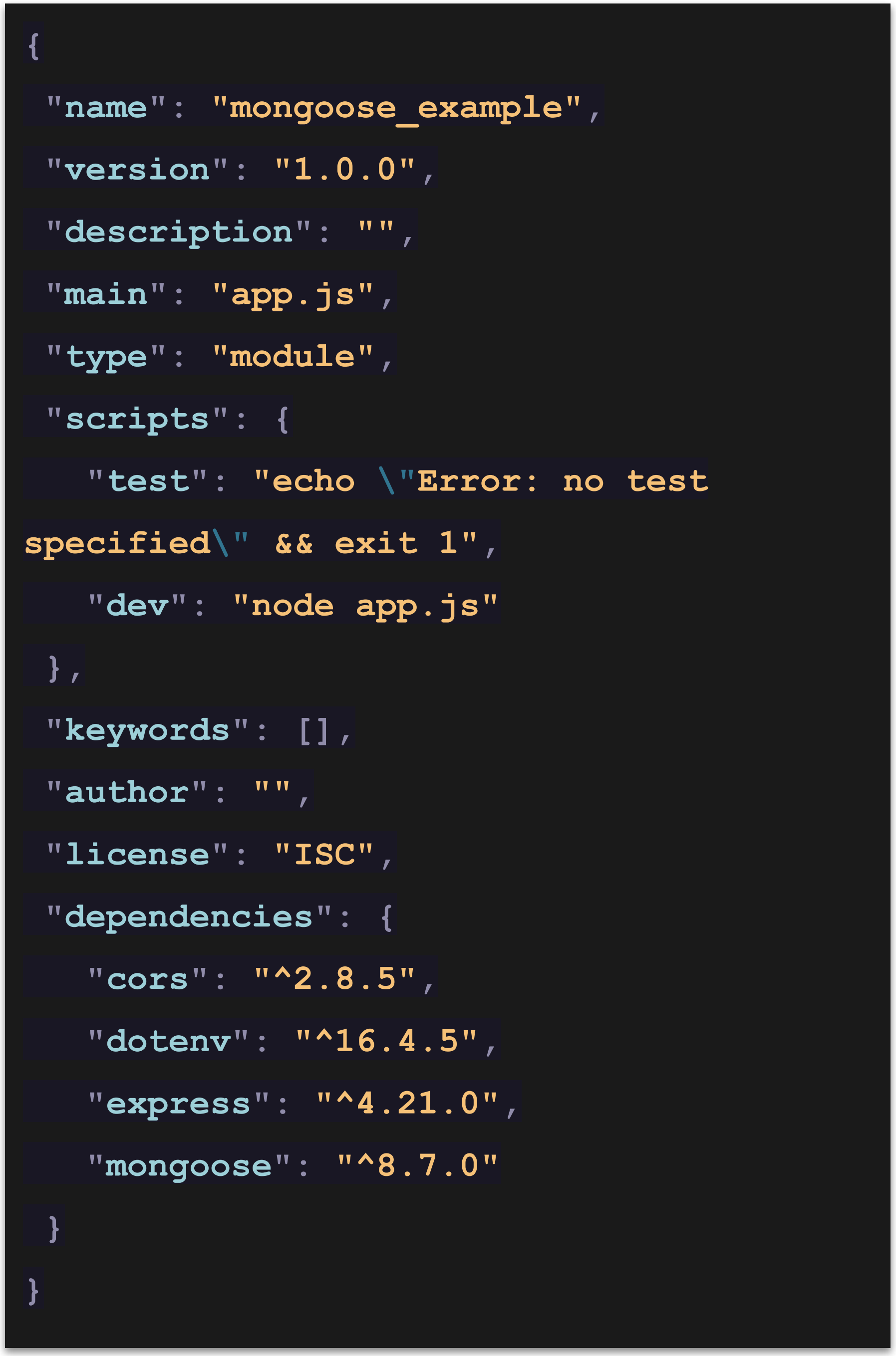
Initial Setup

1. Create a new directory: `mkdir mongoose-example`
2. Cd into new directory: `cd mongoose-example`
3. Initialize a new Node.js project: `npm init-y`
4. Install dependencies: Remember you can install all dependencies at the same time with the **npm i** command followed by your dependencies.

   ```
   npm i express mongoose dotenv
   ```

Server Setup: Changes to package.json

- Make sure the value of the "`main`" field matches what you've named your "entry" file. In this example our entry file is `app.js`
- Add the field "`type`" and give it the value "`module`". This allows us to use import syntax.
- In the "`scripts`" field add a new key value pair for running your server "`dev`": "`node app.js`"



```
{
 "name": "mongoose_example",
 "version": "1.0.0",
 "description": "",
 "main": "app.js",
 "type": "module",
 "scripts": {
   "test": "echo \"Error: no test
specified\" && exit 1",
   "dev": "node app.js"
 },
 "keywords": [],
 "author": "",
 "license": "ISC",
 "dependencies": {
   "cors": "^2.8.5",
   "dotenv": "^16.4.5",
   "express": "^4.21.0",
   "mongoose": "^8.7.0"
 }
}
```

Upright Education

Server Setup: `.env` And `.gitignore`

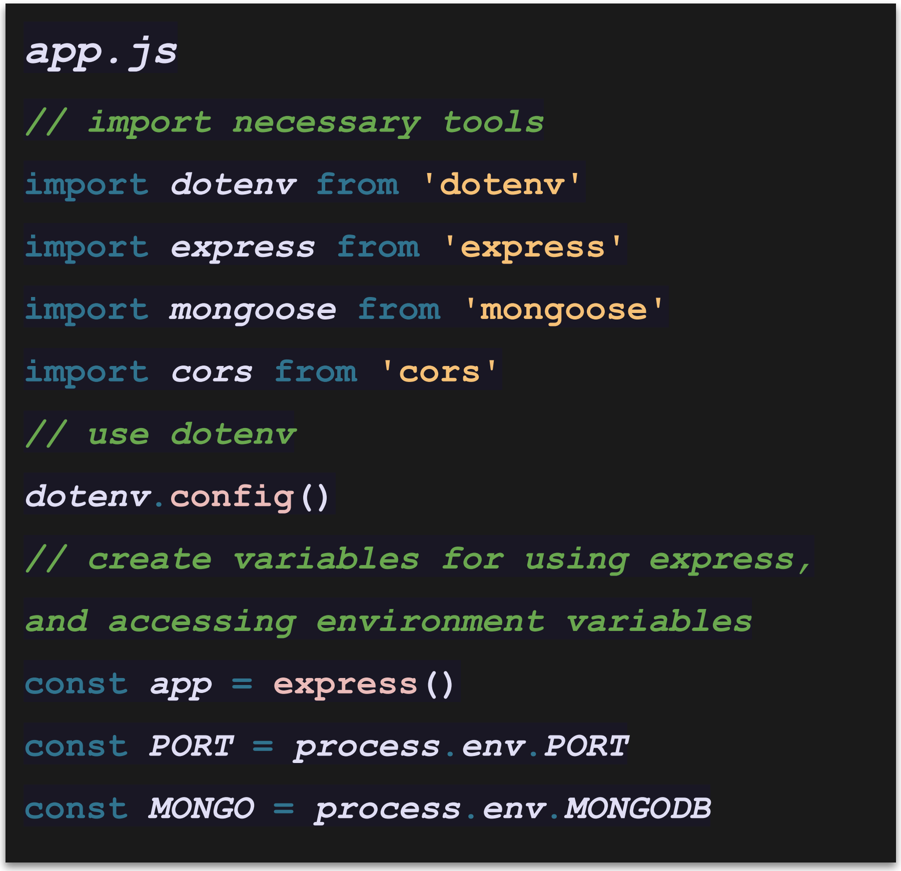
- At the root level of the project create a `.env` file and a `.gitignore` file
- Add environment variables to `.env` file
- Add `.env` file to `.gitignore` file.

```
.env file
PORT = 3000
MONGODB = mongodb://localhost:27017
```

```
.gitignore file
# Add .env file to gitignore
.env
```

Server Setup: `app.js` Importing Dependencies

- Start by importing necessary dependencies
- `dotenv.config()` allows us to access the values in the `.env` file
- Create variables for using express and accessing our environment variables



```
app.js
// import necessary tools
import dotenv from 'dotenv'
import express from 'express'
import mongoose from 'mongoose'
import cors from 'cors'
// use dotenv
dotenv.config()
// create variables for using express,
and accessing environment variables
const app = express()
const PORT = process.env.PORT
const MONGO = process.env.MONGODB
```

Upright Education

Server Setup: Connect Database And Add Middleware

- Use `mongoose.connect()` to establish a connection to the database
- Set that connection to a variable **db**
- Use the **.once()** method to check the connection
- Add express middleware
  - `express.json()` allows processing of json requests
  - **cors()** allows for “cross origin requests”
- Use the `.listen()` method to check what port is listening

```
app.js
// Use the mongoose .connect() method to
connect to database. NOTICE we use the MONGO
variable we defined previously followed by a
"/<name of database>"
mongoose.connect(`${MONGO}/myMongooseTest`)
// Set the connection equal to a variable
"db"
const db = mongoose.connection
// console log to let us know the db is
connected
db.once("open", () =>
console.log(`connected: ${MONGO}`))
// Express middleware
app.use(express.json())
app.use(cors())
// console log to let us know we are
listening on the PORT we defined
app.listen(PORT, () => console.log(`student
server on port: ${PORT}`))
```

Basic Setup Complete

- At this point you should be able to start your server and connect to your database!
    - Run the command `npm run dev`

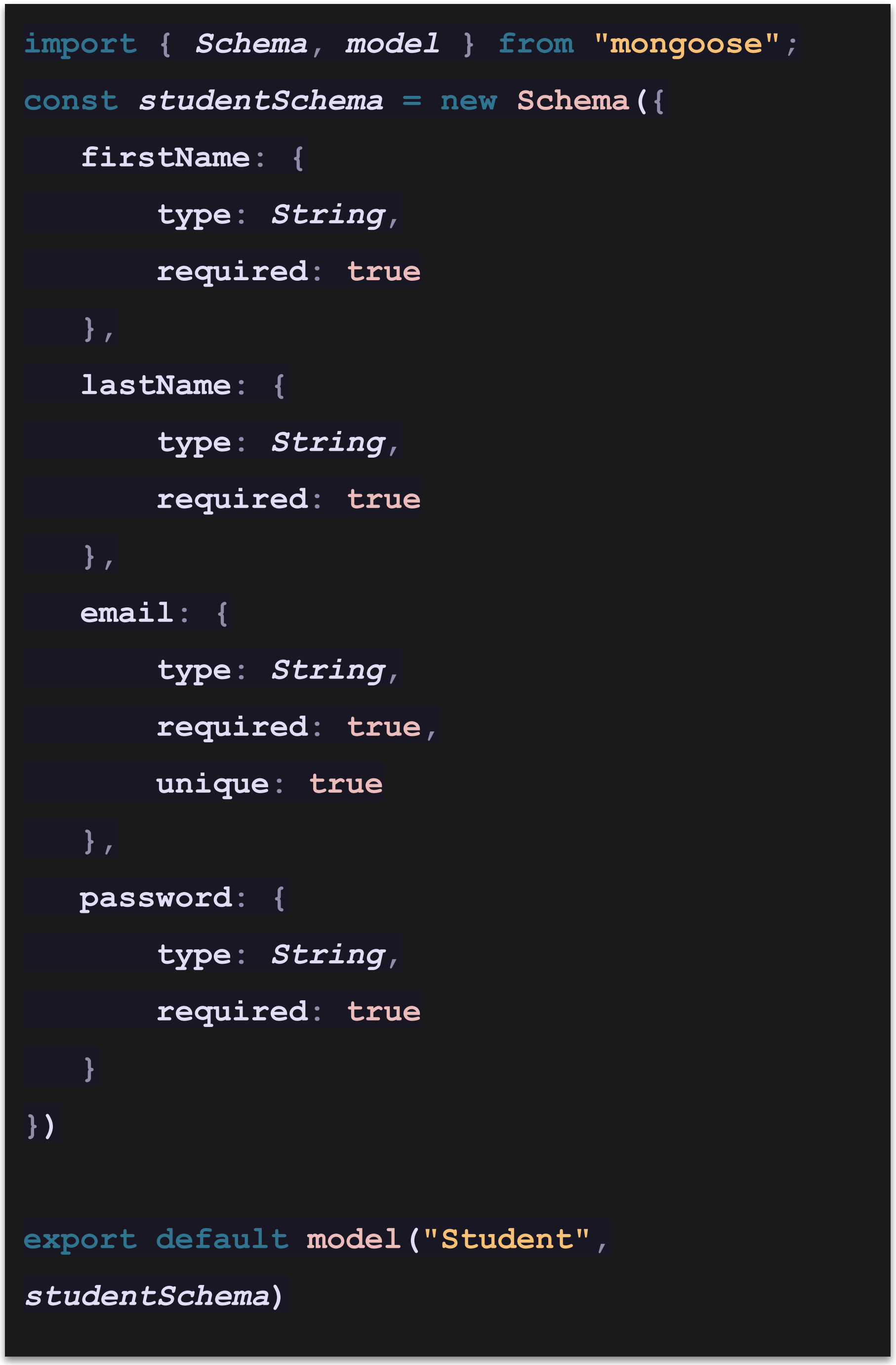
Mongoose Model Setup

- Create a new folder **models** this is where all your models will be stored.
- In this example we use a “Student” as our model

Upright Education

Student Model

- Within the **models** folder create a file **student.model.js**
- Using import syntax we can bring in only the parts of mongoose we need, **Schema**, and **model**



```
import { Schema, model } from "mongoose";
const studentSchema = new Schema({
    firstName: {
        type: String,
        required: true
    },
    lastName: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true
    }
})

export default model("Student",
studentSchema)
```

Mongoose Route Setup

- Create a new folder called **Routes** This is where all routes will be stored.
- The first routes we create will be for our **Student** model so we can name this file **student.route.js**

Upright Education

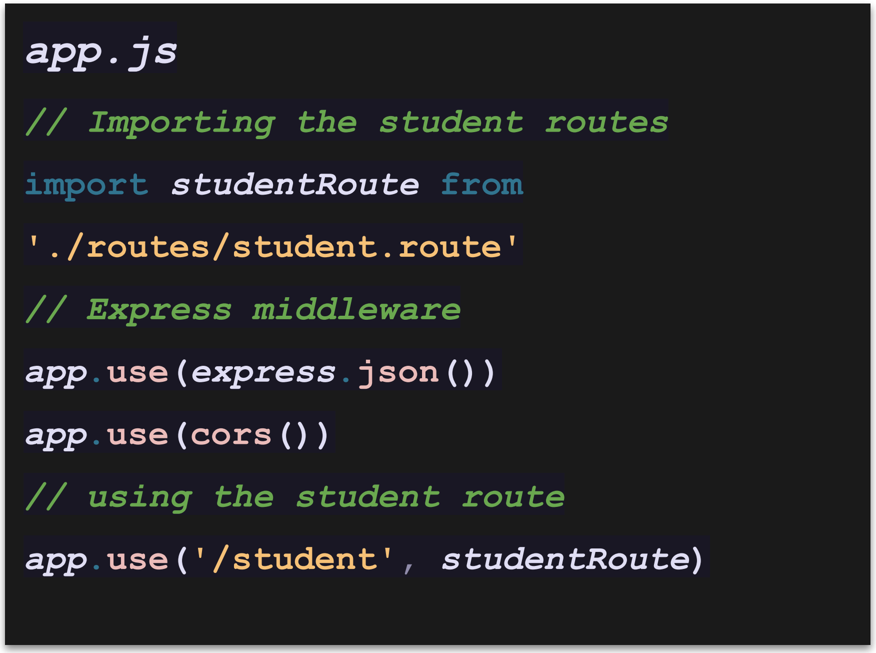
Student Routes

- Import express **Router** using import syntax
- Import the **Student** model we created
- Save the express router functionality to a variable called **router**
- Define a **post** route at **/new** that creates a new student.
- Use the **.save()** method to save the new Student to the database
- Export the **router**

```
import { Router } from "express";
import Student from
'../models/student.model.js'
const router = Router()
router.post('/new', async(req, res) => {
    try {
        const student = new Student({
            firstName: req.body.firstName,
            lastName: req.body.lastName,
            email: req.body.email,
            password: req.body.password
        })
        const newStudent = await
student.save()
        res.status(200).json({
            student: newStudent,
            message: `success`
        })
    } catch (error) {
        console.log(error.message)
    }
})
export default router
```

Adding Routes to `app.js`

- In **app.js** at the bottom of our imports we import `studentRoute`
- After our express middleware we use the `studentRoute` at /students



```
app.js
// Importing the student routes
import studentRoute from
'./routes/student.route'
// Express middleware
app.use(express.json())
app.use(cors())
// using the student route
app.use('/student', studentRoute)
```

Testing With Postman

- Open Postman and test the route you created.
- Make sure:
  - Your server is running and your code has been saved
  - Double check the route when testing in postman
  - Double check the request type (GET, POST, PUT, DELETE) in Postman.