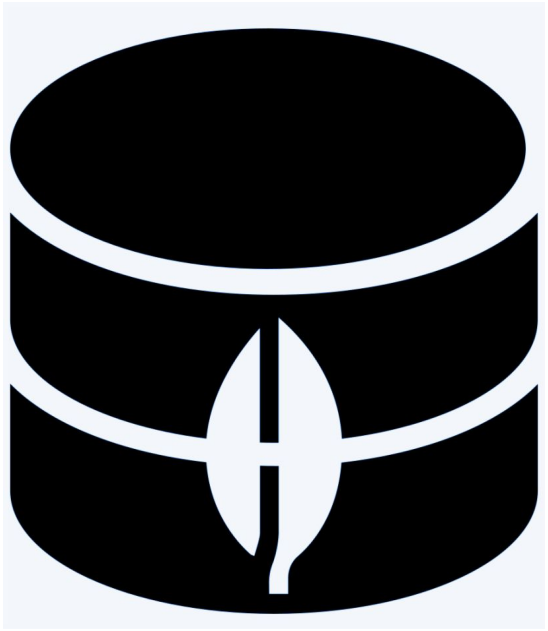


MongoDB: Advanced Queries

SENG 4640
Software Engineering for Web Apps
Winter 2023

Sina Keshvadi
Thompson Rivers University

Review

- **MongoDB** is a NoSQL Database that stores **collections** of **documents**

- Once we've defined a **Schema** we can use the `find` function to select all documents in a collection, or pass a query object to select only certain ones

- Once we have a document, we can update it using the `save` function

We can store data as objects rather than in tables of columns and rows.

We can store data as objects rather than in tables of columns and rows.

What that allows us to do is store objects that have other objects as properties.

We can store data as objects rather than in tables of columns and rows.

What that allows us to do is store objects that have other objects as properties.

For example, you might have a book.

We can store data as objects rather than in tables of columns and rows.

What that allows us to do is store objects that have other objects as properties.

For example, you might have a book.
We can represent this in our database.
It has a title, it has a publication year but it also has authors and each of those authors could be an object as well.

We can store data as objects rather than in tables of columns and rows.

What that allows us to do is store objects that have other objects as properties.

For example, you might have a book.
We can represent this in our database.
It has a title, it has a publication year but it also has authors and each of those authors could be an object as well.

Each author has a name, an affiliation, etc.

In this lecture, we'll start to look at how can we represent these more complicated objects in our database, and then how can we get them out using more advanced queries than we saw before!

```
{
   title: 'Introduction to Algorithms',
   year: 1990,
   authors: [
       {   name: 'Thomas Cormen', affiliation: 'Dartmouth'
       },
       {   name: 'Charles  Leiserson', affiliation: 'MIT'
       },
       {   name: 'Ronald Rivest', affiliation: 'MIT'
       },
       {   name: 'Clifford Stein', affiliation: 'Columbia'
       } ]
}
```

```
{
   title: 'Principles of Compiler Design',
   year: 1977,
   authors: [
       {   name: 'Alfred Aho', affiliation: 'Bell Labs'
       },
       {   name: 'Jeffrey Ullman', affiliation: 'Princeton'
       }
   ]
}
```

JSON representation of two sample books

How to set up the schema in our Mongo database so that we can use it in the rest of the app?

```
// book.js
var mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/myDatabase");
var Schema = mongoose.Schema;

var authorSchema = new Schema({ name: String, affiliation: String });

var bookSchema = new Schema({
 title: { type: String, required: true, unique: true },
 year: Number,
 authors: [authorSchema],
});
module.exports = mongoose.model("Book", bookSchema);
```

book.js

```
// book.js
var mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/myDatabase");
var Schema = mongoose.Schema;

var authorSchema = new Schema({ name: String, affiliation: String });

var bookSchema = new Schema({
 title: { type: String, required: true, unique: true },
 year: Number,
 authors: [authorSchema],
});
module.exports = mongoose.model("Book", bookSchema);
```

You can specify any name that you'd like here.

```
// book.js
var mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/myDatabase");
var Schema = mongoose.Schema;

var authorSchema = new Schema({ name: String, affiliation: String });

var bookSchema = new Schema({
 title: { type: String, required: true, unique: true },
 year: Number,
 authors: [authorSchema],
});
module.exports = mongoose.model("Book", bookSchema);
```

Then we'll get the defaults or base schema object.
Then we'll create our author schema.

```
// book.js
var mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/myDatabase");
var Schema = mongoose.Schema;

var authorSchema = new Schema({ name: String, affiliation: String });

var bookSchema = new Schema({
 title: { type: String, required: true, unique: true },
 year: Number,
 authors: [authorSchema],
});
module.exports = mongoose.model("Book", bookSchema);
```

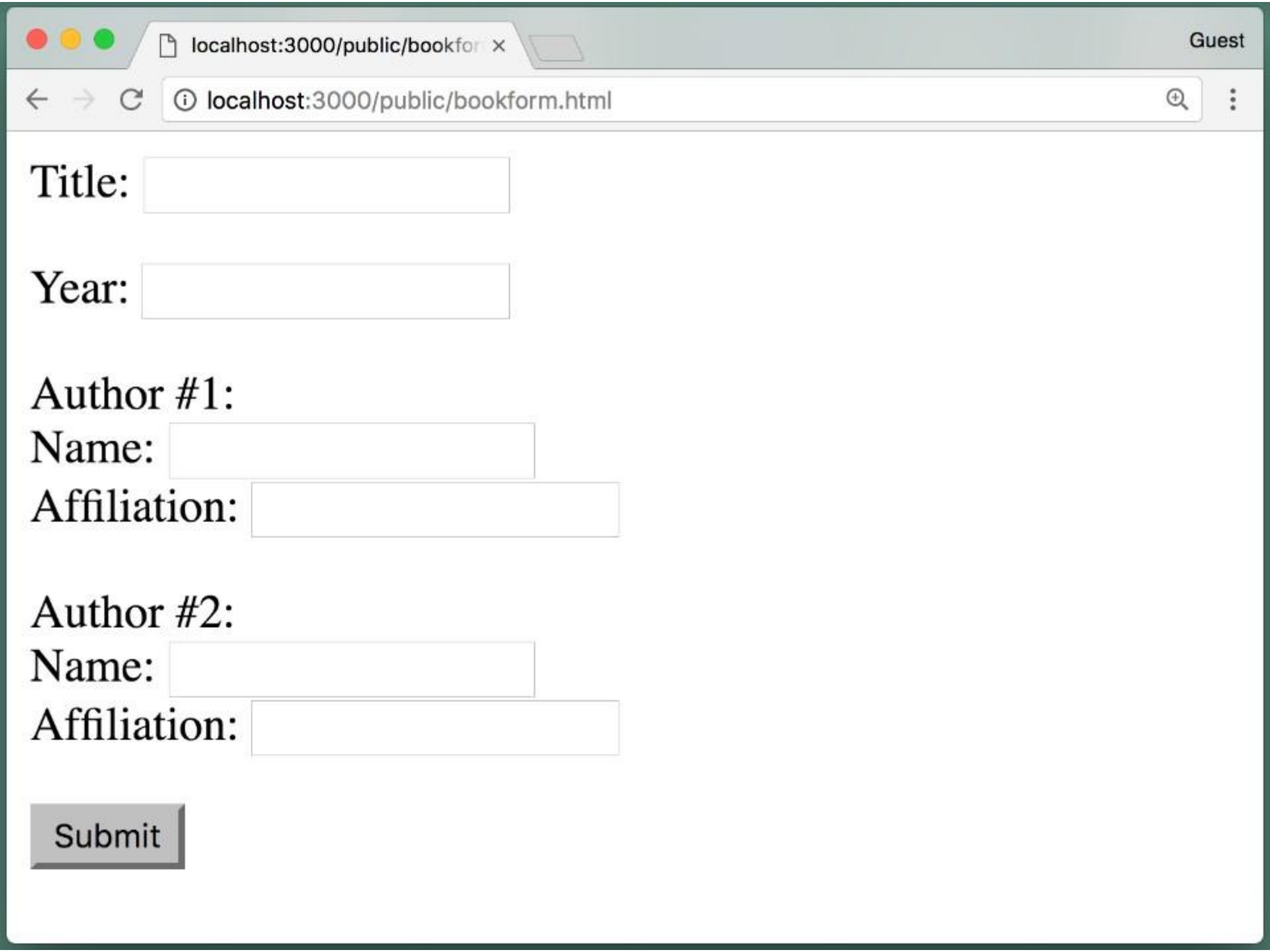
Then we'll create our book schema.

```
// book.js
var mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/myDatabase");
var Schema = mongoose.Schema;

var authorSchema = new Schema({ name: String, affiliation: String });

var bookSchema = new Schema({
 title: { type: String, required: true, unique: true },
 year: Number,
 authors: [authorSchema],
});
module.exports = mongoose.model("Book", bookSchema);
```

authors is an array of authorSchema objects.
This allows us to easily create objects that have other objects as properties without even needing to know in advance how many authors each book will have.

```
// book.js
var mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/myDatabase");
var Schema = mongoose.Schema;

var authorSchema = new Schema({ name: String, affiliation: String });

var bookSchema = new Schema({
 title: { type: String, required: true, unique: true },
 year: Number,
 authors: [authorSchema],
});
module.exports = mongoose.model("Book", bookSchema);
```

export the class or the model that we want to use in the rest of the app.
we're only going to be using books and not authors.


localhost:3000/public/bookfor
Guest
localhost:3000/public/bookform.html
Title:
Year:
Author #1:
Name:
Affiliation:
Author #2:
Name:
Affiliation:
Submit

We could use some client-side JavaScript to dynamically generate the form depending on the number of authors the book will have.

```
<!-- book.html -->
<form action='/createbook' method='post'>
    <p> Title: <input name='title'> </p>
    Year: <input name='year'>
    <p>
        Author #1: <br>
        Name: <input name='authors[0][name]'> <br> Affiliation:
        <input name='authors[0][affiliation]'>
    </p>
    <p>
        Author #2: <br>
        Name: <input name='authors[1][name]'> <br> Affiliation:
        <input name='authors[1][affiliation]'>
    </p>
    <p> <input type='submit' value='Submit'> </p>
</form>
```

```
<!-- book.html -->
<form action='/createbook' method='post'>
    <p> Title: <input name='title'> </p>
    Year: <input name='year'>
    <p>
        Author #1: <br>
        Name: <input name='authors[0][name]'> <br> Affiliation:
        <input name='authors[0][affiliation]'>
    </p>
    <p>
        Author #2: <br>
        Name: <input name='authors[1][name]'> <br> Affiliation:
        <input name='authors[1][affiliation]'>
    </p>
    <p> <input type='submit' value='Submit'> </p>
</form>
```

book.html

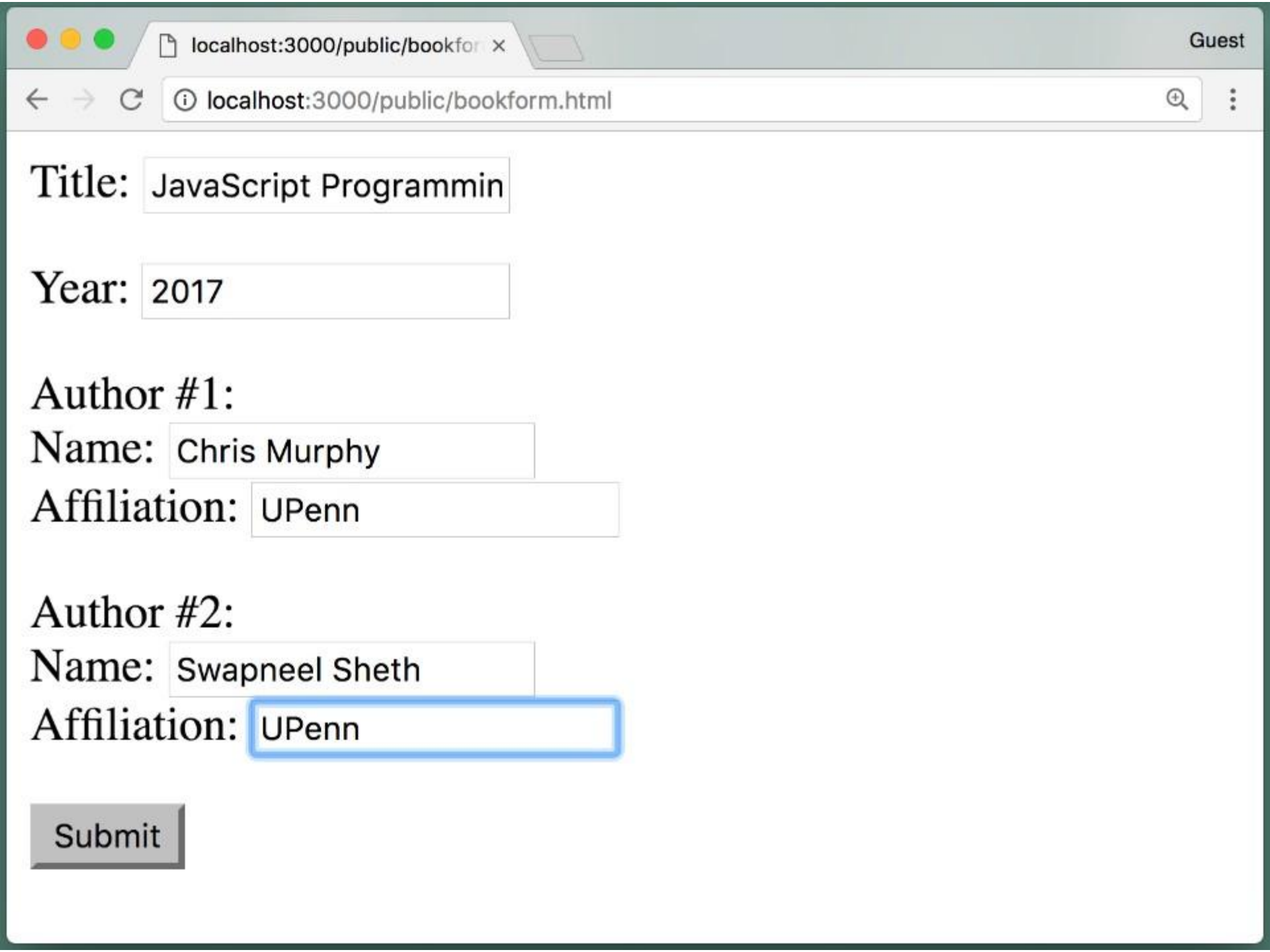
```
<!-- book.html -->
<form action='/createbook' method='post'>
    <p> Title: <input name='title'> </p>
    Year: <input name='year'>
    <p>
        Author #1: <br>
        Name: <input name='authors[0][name]'> <br> Affiliation:
        <input name='authors[0][affiliation]'>
    </p>
    <p>
        Author #2: <br>
        Name: <input name='authors[1][name]'> <br> Affiliation:
        <input name='authors[1][affiliation]'>
    </p>
    <p> <input type='submit' value='Submit'> </p>
</form>
```

```
<!-- book.html -->
<form action='/createbook' method='post'>
    <p> Title: <input name='title'> </p>
    Year: <input name='year'>
    <p>
        Author #1: <br>
        Name: <input name='authors[0][name]'> <br> Affiliation:
        <input name='authors[0][affiliation]'>
    </p>
    <p>
        Author #2: <br>
        Name: <input name='authors[1][name]'> <br> Affiliation:
        <input name='authors[1][affiliation]'>
    </p>
    <p> <input type='submit' value='Submit'> </p>
</form>
```

Because we want on the server this entire object to contain multiple authors, and those authors to be in an array, and those authors to have properties such as name and affiliation, we can use this syntax here.
This says that the name of this input field is going to be part of some array. And in particular is going to be a property of some element that's in an array.

```
<!-- book.html -->
<form action='/createbook' method='post'>
   <p> Title: <input name='title'> </p>
   Year: <input name='year'>
   <p>
       Author #1: <br>
       Name: <input name='authors[0][name]'> <br> Affiliation:
       <input name='authors[0][affiliation]'>
   </p>
   <p>
       Author #2: <br>
       Name: <input name='authors[1][name]'> <br> Affiliation:
       <input name='authors[1][affiliation]'>
   </p>
   <p> <input type='submit' value='Submit'> </p>
</form>
```

```
<!-- book.html -->
<form action='/createbook' method='post'>
   <p> Title: <input name='title'> </p>
   Year: <input name='year'>
   <p>
       Author #1: <br>
       Name: <input name='authors[0][name]'> <br> Affiliation:
       <input name='authors[0][affiliation]'>
   </p>
   <p>
       Author #2: <br>
       Name: <input name='authors[1][name]'> <br> Affiliation:
       <input name='authors[1][affiliation]'>
   </p>
   <p> <input type='submit' value='Submit'> </p>
</form>
```


localhost:3000/public/bookfor
Guest
localhost:3000/public/bookform.html
Title: JavaScript Programmin
Year: 2017
Author #1:
Name: Chris Murphy
Affiliation: UPenn
Author #2:
Name: Swapneel Sheth
Affiliation: UPenn
Submit

```
// index.js
var express = require("express");
var app = express();
app.set("view engine", "ejs");
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({  extended: true }));

var Book = require("./Book.js");

app.use("/createbook", async (req, res) => {
 try {
   const newBook = new Book(req.body);
   await newBook.save();
   res.render("created", { book: newBook });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
});
```

Here, we have the server-side code.
This is our Node Express app that's on our server.

```
// index.js
var express = require("express");
var app = express();
app.set("view engine", "ejs");
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));

var Book = require("./Book.js");

app.use("/createbook", async (req, res) => {
 try {
   const newBook = new Book(req.body);
   await newBook.save();
   res.render("created", { book: newBook });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
});
```

require book.js which will allow us to import the book model or book class that we're going to use when we write data to our database.

```
// index.js
var express = require("express");
var app = express();
app.set("view engine", "ejs");
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));

var Book = require("./Book.js");

app.use("/createbook", async (req, res) => {
 try {
   const newBook = new Book(req.body);
   await newBook.save();
   res.render("created", { book: newBook });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
});
```

create the route for /createbook

```
// index.js
var express = require("express");
var app = express();
app.set("view engine", "ejs");
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));

var Book = require("./Book.js");

app.use("/createbook", async (req, res) => {
 try {
   console.log(req.body);
   const newBook = new Book(req.body);
   await newBook.save();
   res.render("created", { book: newBook });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }});
```

log the request body to see what does the body of the request look like after we submit the form that we saw in the previous slide.

```
{ title: 'JavaScript Programming',
  year: '2017',
  authors:
   [ { name: 'Chris Murphy', affiliation: 'UPenn' },
     { name: 'Swapneel Sheth', affiliation: 'UPenn' } ] }
```

```
var authorSchema = new Schema({
      name: String,
      affiliation: String
   });

var bookSchema = new Schema({
      title: {type: String, required: true, unique: true},
      year: Number,
      authors: [authorSchema]
   });
```

```
// index.js
var express = require("express");
var app = express();
app.set("view engine", "ejs");
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));

var Book = require("./Book.js");

app.use("/createbook", async (req, res) => {
 try {
   const newBook = new Book( req.body);
   await newBook.save();
   res.render("created", { book: newBook });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
});
```

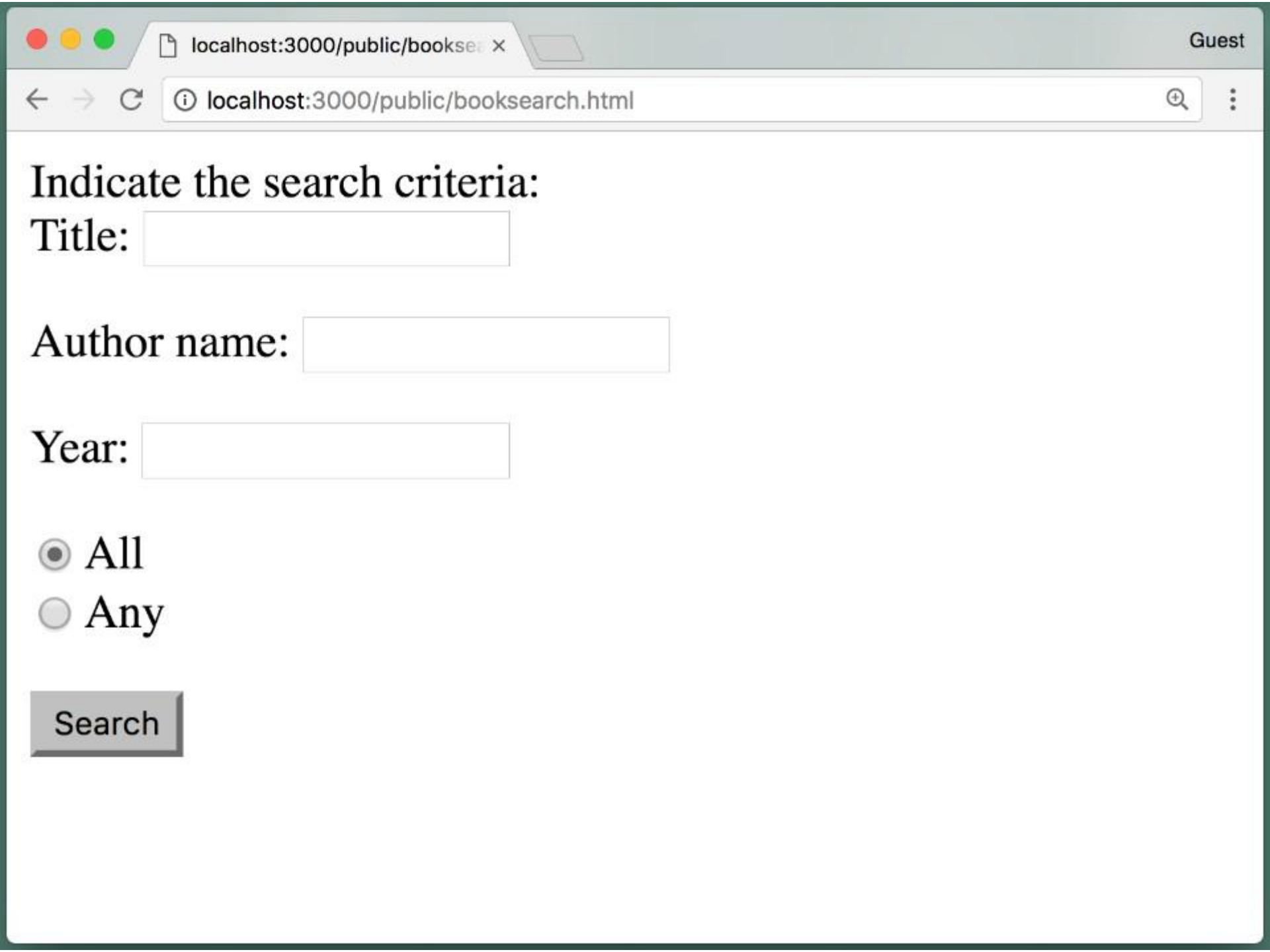
use req.body to initialize the object that we're creating

```
// index.js
var express = require("express");
var app = express();
app.set("view engine", "ejs");
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));

var Book = require("./Book.js");

app.use("/createbook", async (req, res) => {
 try {
   const newBook = new Book(req.body);
   await newBook.save();
   res.render("created", { book: newBook });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
});
```

```
// index.js
var express = require("express");
var app = express();
app.set("view engine", "ejs");
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));

var Book = require("./Book.js");

app.use("/createbook", async (req, res) => {
 try {
   const newBook = new Book(req.body);
   await newBook.save();
   res.render("created", { book: newBook });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
});
```

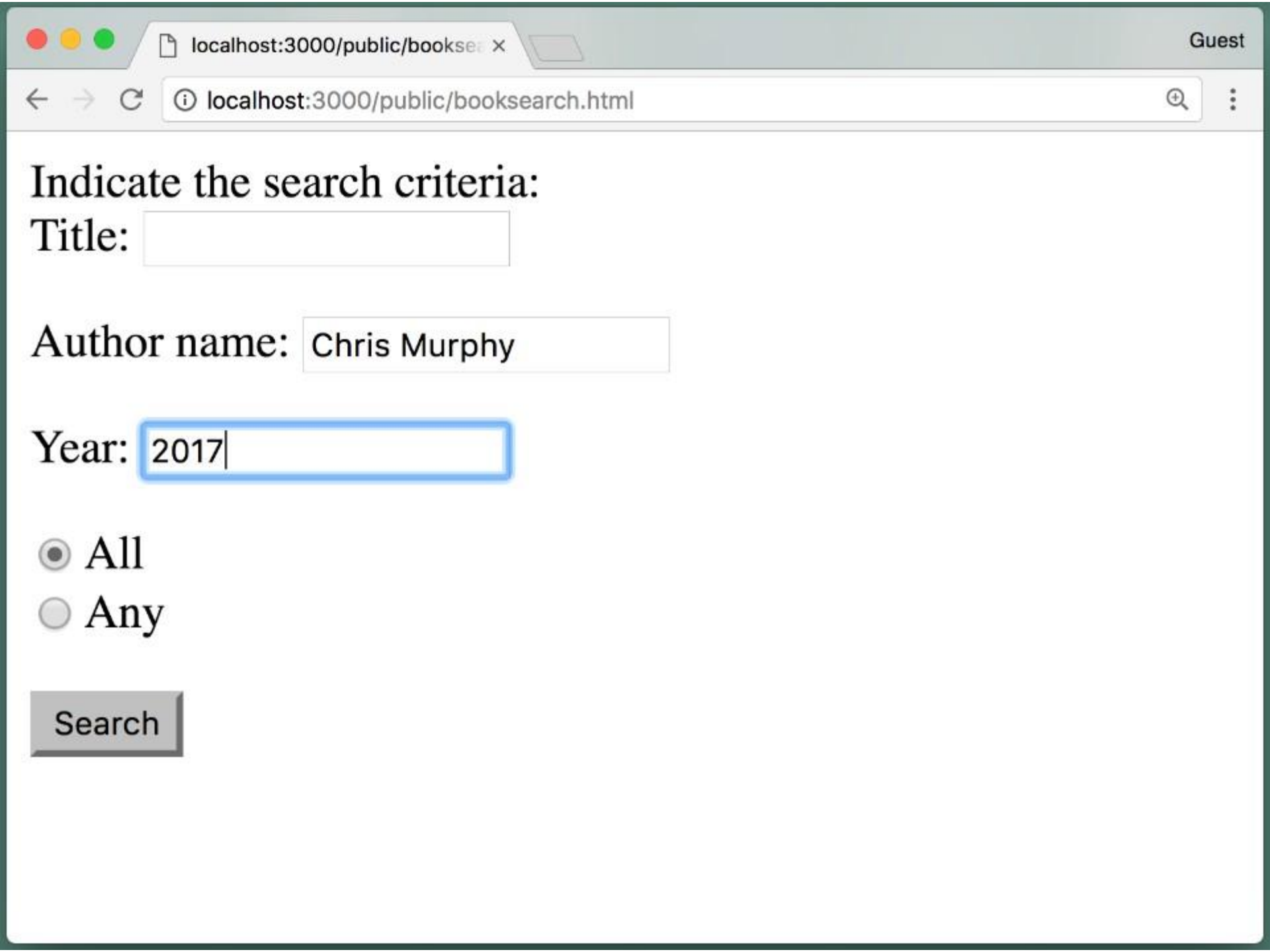
we don’t show the created.ejs in this lecture.

In the next example, we're going to see how can we do queries using the object which is the property of some other object to get data out of the database.


localhost:3000/public/booksea
Guest
localhost:3000/public/booksearch.html
Indicate the search criteria:
Title:
Author name:
Year:
All
Any
Search

```
// booksearch.html
<form action="search" method="post">
        Title: <input name="title">
    <p>
        Author name: <input name="name">
    </p>
    <p>
        Year: <input name="year">
    </p>
    <p>
        <input type="radio" name="which" value="all" checked="">All<br>
        <input type="radio" name="which" value="any">Any<br>
    </p>
    <p>
        <input type="submit" value="Search">
    </p>
</form>
```

checked means that, by default, when the page is rendered, this radio button is selected.


localhost:3000/public/booksea
Guest
localhost:3000/public/booksearch.html
Indicate the search criteria:
Title:
Author name:
Chris Murphy
Year:
2017
All
Any
Search

```
// index.js
app.use("/search", (req, res) => {
 if (req.body.which == "all") {
   searchAll(req, res);
 } else if (req.body.which == "any") {
   searchAny(req, res);
 } else {
   searchAll(req, res);
 }
});
```

Because doing the all search and the any search are different enough, we can put them in two separate functions.

searchAll function

```
// index.js
async function searchAll(req, res) {
 const query = {};
 if (req.body.title) {
   query.title = req.body.title;
 }
 if (req.body.name) {
   query["authors.name"] = req.body.name;
 }
 if (req.body.year) {
   query.year = req.body.year;
 }

 try {
   const books = await Book.find(query);
   res.render("books", { books });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
}
```

```
// index.js
async function searchAll(req, res) {
 const query = {};
 if (req.body.title) {
   query.title = req.body.title;
 }
 if (req.body.name) {
   query["authors.name"] = req.body.name;
 }
 if (req.body.year) {
   query.year = req.body.year;
 }

 try {
   const books = await Book.find(query);
   res.render("books", { books });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
}
```

```
// index.js
async function searchAll(req, res) {
 const query = {};
 if (req.body.title) {
   query.title = req.body.title;
 }
 if (req.body.name) {
   query["authors.name"] = req.body.name;
 }
 if (req.body.year) {
   query.year = req.body.year;
 }

 try {
   const books = await Book.find(query);
   res.render("books", { books });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
}
```

This object (query) is going to be used to conduct the query in our database.

```
// index.js
async function searchAll(req, res) {
 const query = {};
 if (req.body.title) {
   query.title = req.body.title;
 }
 if (req.body.name) {
   query["authors.name"] = req.body.name;
 }
 if (req.body.year) {
   query.year = req.body.year;
 }

 try {
   const books = await Book.find(query);
   res.render("books", { books });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
}
```

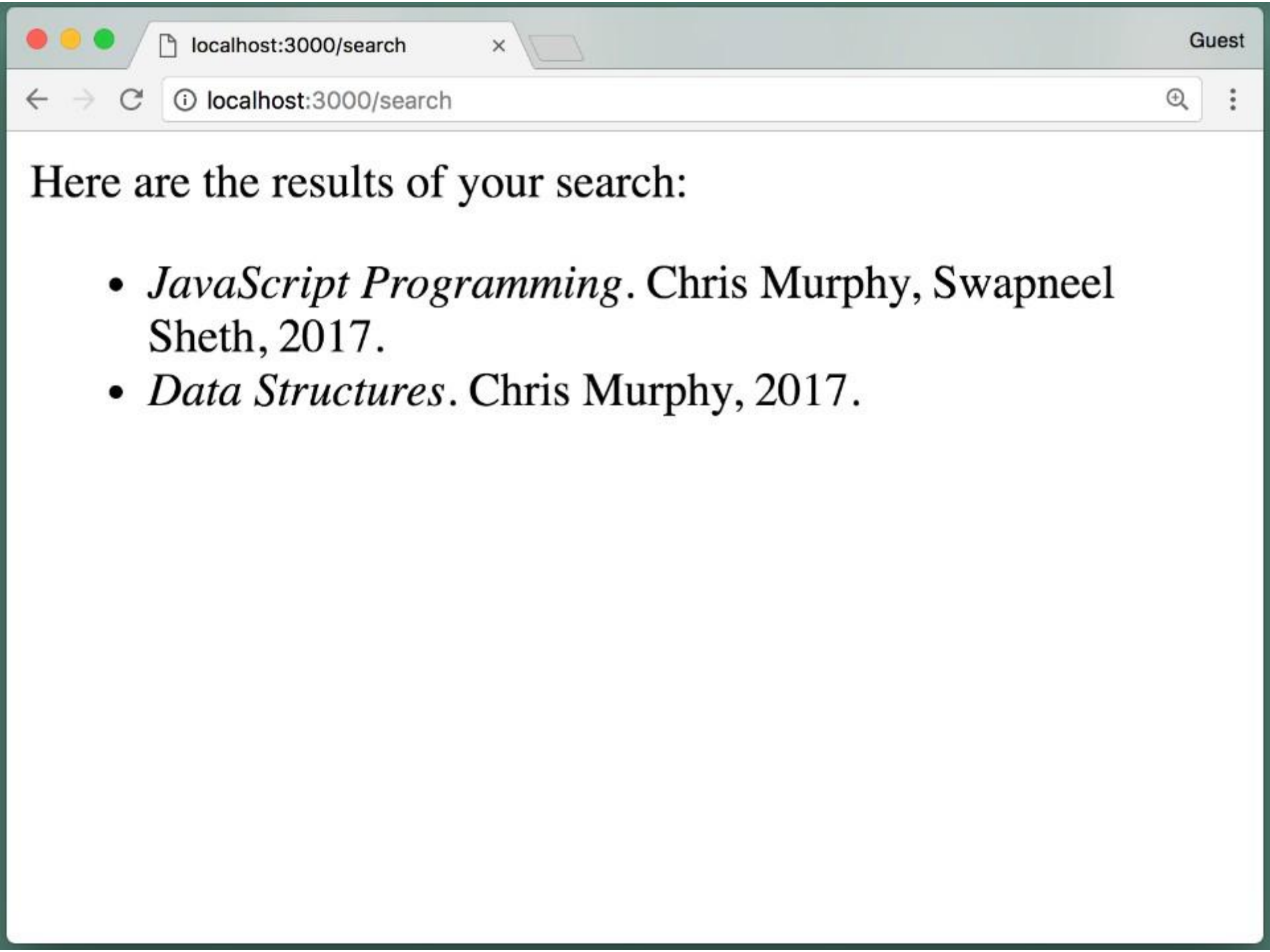
```
// index.js
async function searchAll(req, res) {
 const query = {};
 if (req.body.title) {
   query.title = req.body.title;
 }
 if (req.body.name) {
   query["authors.name"] = req.body.name;
 }
 if (req.body.year) {
   query.year = req.body.year;
 }

 try {
   const books = await Book.find(query);
   res.render("books", { books });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
}
```

Book doesn't have an author name. The book has authors and authors have a name. So, if there is a name in the request body, then we need to use this notation to set the property name in the query object.

In Mongo, if we're going to search for property in an object that is a property of another object, we need to use this notation, for instance, authors.name.

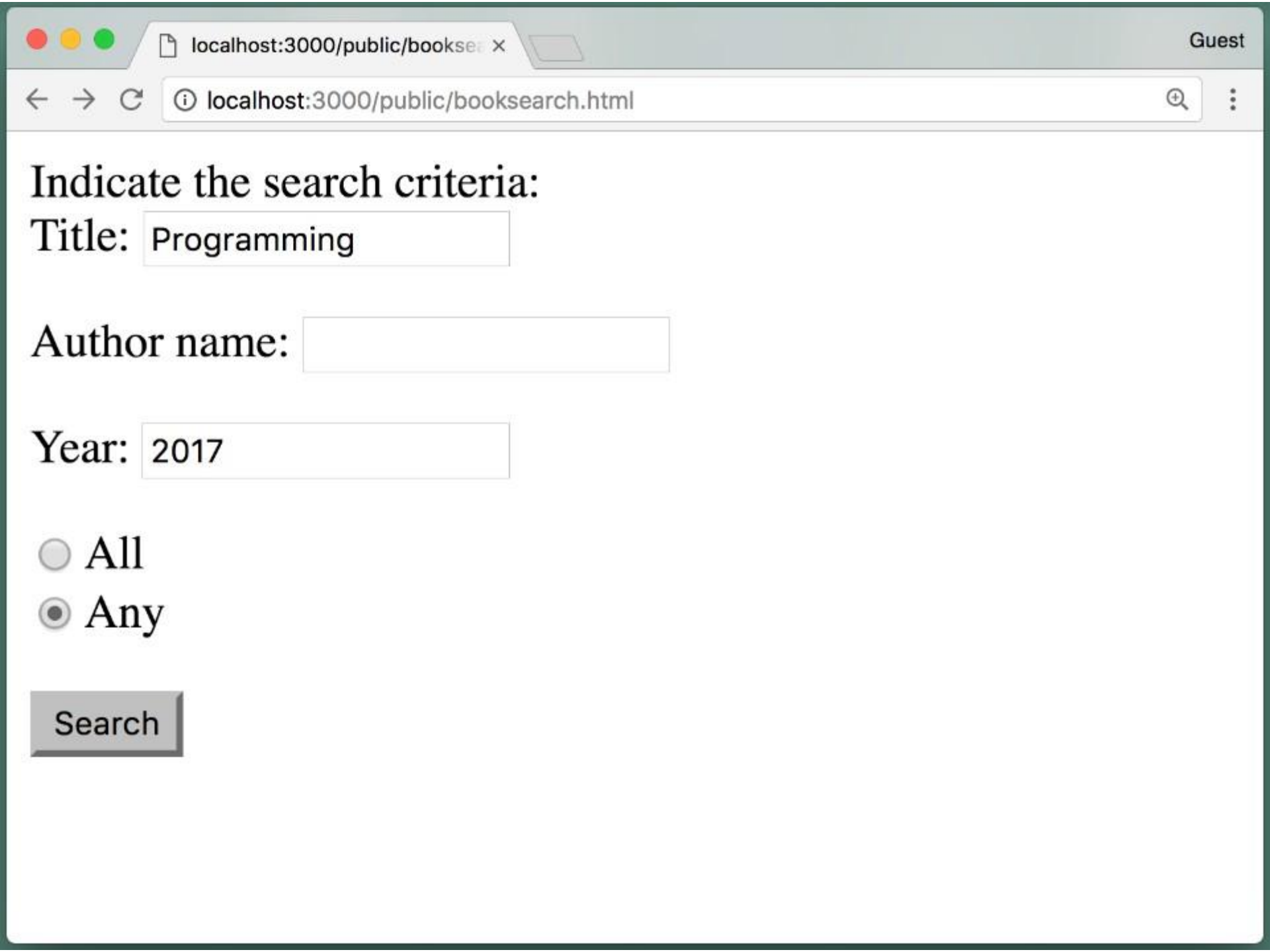
```
// index.js
async function searchAll(req, res) {
 const query = {};
 if (req.body.title) {
   query.title = req.body.title;
 }
 if (req.body.name) {
   query["authors.name"] = req.body.name;
 }
 if (req.body.year) {
   query.year = req.body.year;
 }

 try {
   const books = await Book.find(query);
   res.render("books", { books });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
}
```

```
// index.js
async function searchAll(req, res) {
 const query = {};
 if (req.body.title) {
   query.title = req.body.title;
 }
 if (req.body.name) {
   query["authors.name"] = req.body.name;
 }
 if (req.body.year) {
   query.year = req.body.year;
 }

 try {
   const books = await Book.find(query);
   res.render("books", { books });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
}
```


localhost:3000/search
Guest
localhost:3000/search
Here are the results of your search:
• *JavaScript Programming*. Chris Murphy, Swapneel Sheth, 2017.
• *Data Structures*. Chris Murphy, 2017.

```
<!-- This is views/books.ejs -->

Here are the results of your search:
<ul>
   <% books.forEach( (book)=> { %>

       <li>
           <i>
               <%= book.title %>
           </i>.

           <% book.authors.forEach( (author)=> { if (author.name) { %>

               <%= author.name %>,

                   <% } }); %>

                       <%= book.year %>.

       </li>

       <% }); %>

</ul>
```

In this next example, we'll see how we can do an **or** search where we're able to match any of the criteria that we specify in the form.


localhost:3000/public/booksea
Guest
localhost:3000/public/booksearch.html
Indicate the search criteria:
Title: Programming
Author name:
Year: 2017
All
Any
Search

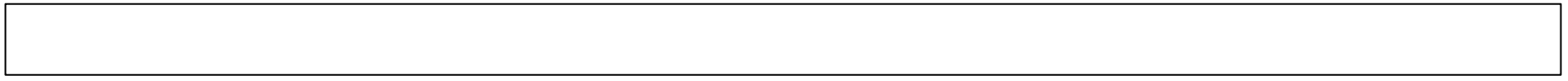
```
// index.js
app.use("/search", (req, res) => {
 if (req.body.which == "all") {
   searchAll(req, res);
 } else if (req.body.which == "any") {
   searchAny(req, res);
 } else {
   searchAll(req, res);
 }
});
```

Because doing the all search and the any search are different enough, we can put them in two separate functions.

```
async function searchAny(req, res) {
 try {
   const terms = [];
   if (req.body.title) {
     terms.push({ title: { title: req.body.title } });
   }
   if (req.body.name) {
     terms.push({ "authors.name": req.body.name });
   }
   if (req.body.year) {
     terms.push({ year: req.body.year });
   }
   const query = { $or: terms };
   const books = await Book.find(query).sort({ title: "asc" });
   res.render("books", { books: books });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }
}
```

The search any function is similar to the search all function, with one big difference about how we construct the query.

```
// index.js
async function searchAny(req, res) {
 try {
   const terms = [];
   if (req.body.title) {
     terms.push({ title: { title: req.body.title } });
   }
   if (req.body.name) {
     terms.push({ "authors.name": req.body.name });
   }
   if (req.body.year) {
     terms.push({ year: req.body.year });
   }
   const query = { $or: terms };
   const books = await Book.find(query).sort({ title: "asc" });
   res.render("books", { books: books });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }}
```



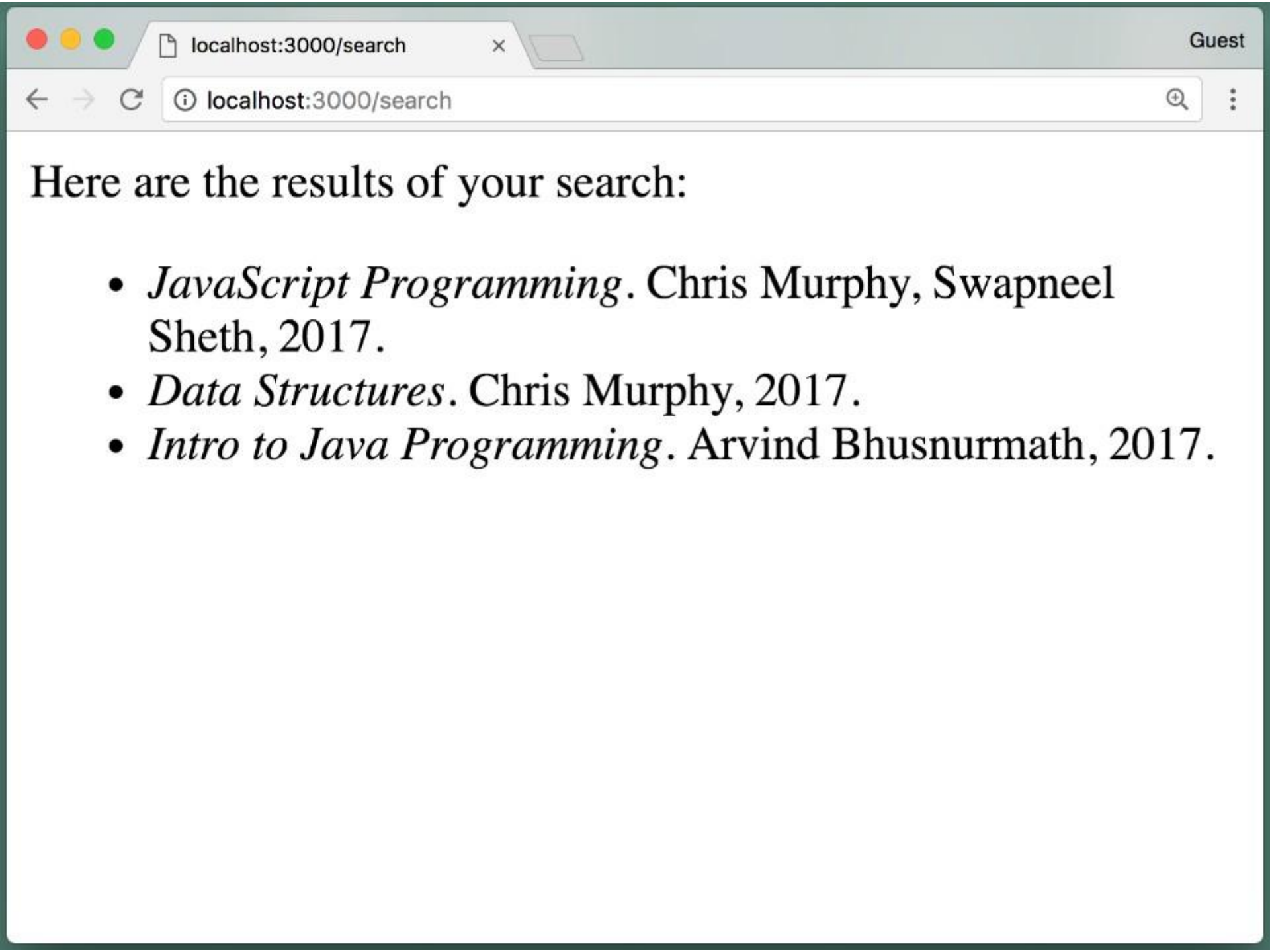
```
// index.js
async function searchAny(req, res) {
 try {
   const terms = [];
   if (req.body.title) {
     terms.push({ title: { title: req.body.title } });
   }
   if (req.body.name) {
     terms.push({ "authors.name": req.body.name });
   }
   if (req.body.year) {
     terms.push({ year: req.body.year });
   }
   const query = { $or: terms };
   const books = await Book.find(query).sort({ title: "asc" });
   res.render("books", { books: books });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }}
```

```
// index.js
async function searchAny(req, res) {
 try {
   const terms = [];
   if (req.body.title) {
     terms.push({ title: { title: req.body.title } });
   }
   if (req.body.name) {
     terms.push({ "authors.name": req.body.name });
   }
   if (req.body.year) {
     terms.push({ year: req.body.year });
   }
   const query = { $or: terms };
   const books = await Book.find(query).sort({ title: "asc" });
   res.render("books", { books: books });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }}
```

This is how we can do an **or** search or an any search in Mongo using a **query object** that has this **property name** and then an array of query objects.
$or is the property name.

```
// index.js
async function searchAny(req, res) {
 try {
   const terms = [];
   if (req.body.title) {
     terms.push({ title: { title: req.body.title } });
   }
   if (req.body.name) {
     terms.push({ "authors.name": req.body.name });
   }
   if (req.body.year) {
     terms.push({ year: req.body.year });
   }
   const query = { $or: terms };
   const books = await Book.find(query)sort({ title: "asc" });
   res.render("books", { books: books });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }}
```

if you want to sort the results, add sort function after the find function.


localhost:3000/search
Guest
localhost:3000/search
Here are the results of your search:
• *JavaScript Programming*. Chris Murphy, Swapneel Sheth, 2017.
• *Data Structures*. Chris Murphy, 2017.
• *Intro to Java Programming*. Arvind Bhusnurmath, 2017.

Mongo will do a search for an exact match.

Mongo will do a search for an exact match.

So how can I tell Mongo that I want to do a search for a title that includes the word like “programming”?

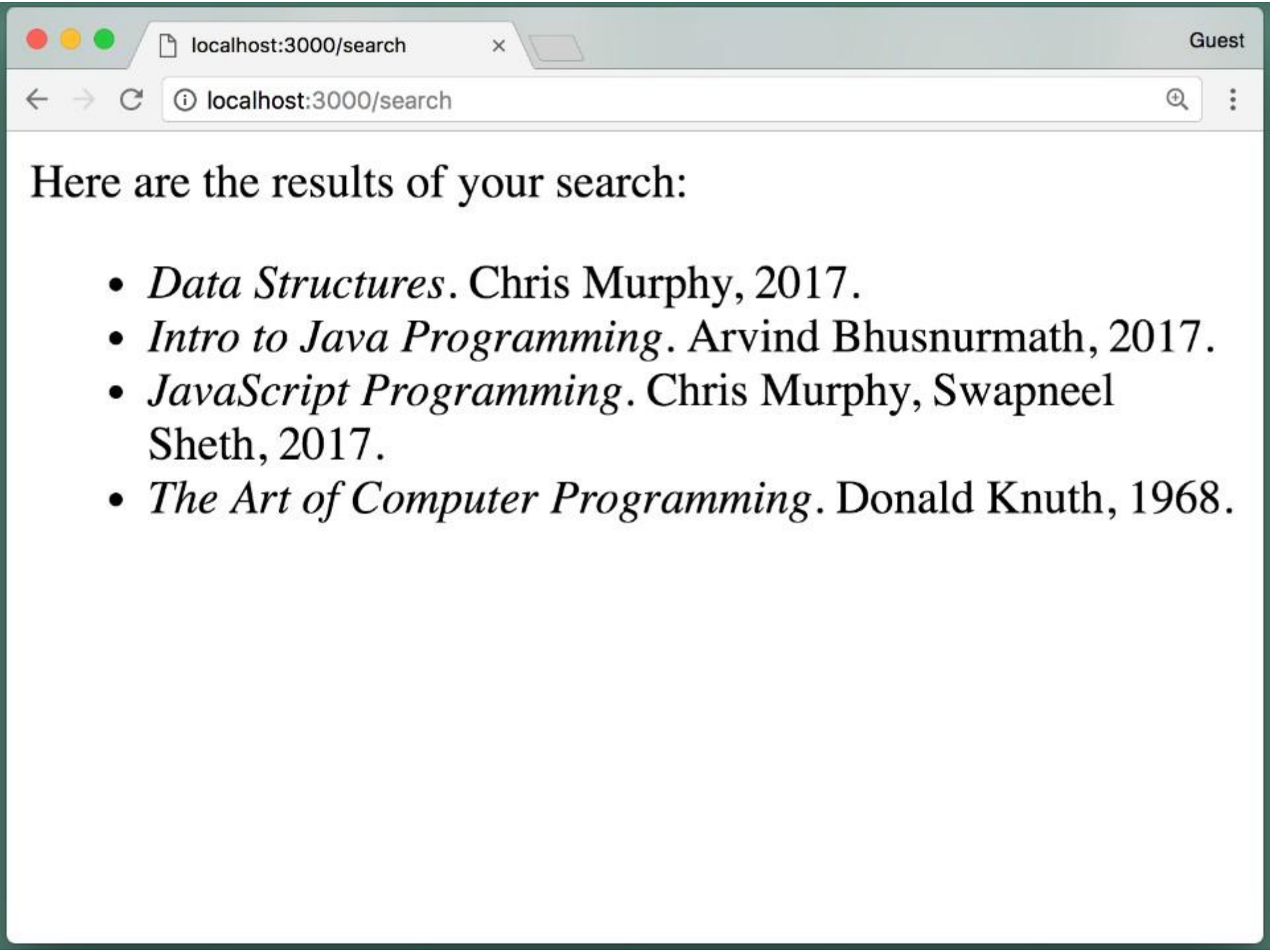
Mongo will do a search for an exact match.

So how can I tell Mongo that I want to do a search for a title that includes the word like “programming”?

The solution is to use a **regular expression.**

```
// index.js
async function searchAny(req, res) {
 try {
   const terms = [];
   if (req.body.title) {
     terms.push({ title: { $regex: req.body.title } });
   }
   if (req.body.name) {
     terms.push({ "authors.name": req.body.name });
   }
   if (req.body.year) {
     terms.push({ year: req.body.year });
   }
   const query = { $or: terms };
   const books = await Book.find(query).sort({ title: "asc" });
   res.render("books", { books: books });
 } catch (err) {
   res.type("html").status(500);
   res.send("Error: " + err);
 }}
```

Here rather than just setting title to request.body.title, we'll set it equal to an object where the key is $regex and then this is the value. This will create a regular expression out of this value here. And when we use it in our query any title that includes this value will match that pattern.



Here are the results of your search:

- *Data Structures*. Chris Murphy, 2017.
- *Intro to Java Programming*. Arvind Bhusnurmath, 2017.
- *JavaScript Programming*. Chris Murphy, Swapneel Sheth, 2017.
- *The Art of Computer Programming*. Donald Knuth, 1968.

Summary

- MongoDB allows us to have a Schema in which one document contains other documents
- We can then do queries for documents using the properties of the documents they contain
- We can also do “all” and “any” queries by passing objects to the `find` function
- And sort the results using `sort`