COSI 152A

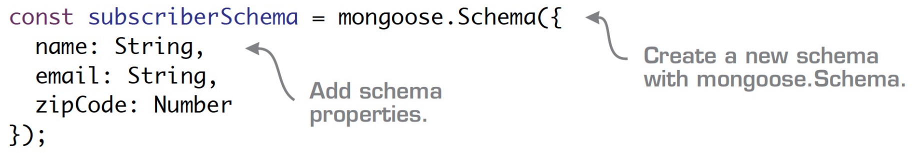
Web Application Development

Main Point Preview

- In this lesson, you will learn to:
  - build the first model and schema
  - connect routes to controllers and to models

Creating a Schema for Subscribers

- mongoose.Schema offers a constructor that allows you to build a schema object with the given parameters.



```
const subscriberSchema = mongoose.Schema({
  name: String,
  email: String,
  zipCode: Number
});
```

- According to this schema someone’s name can’t be a number.

Applying the Schema to a Model

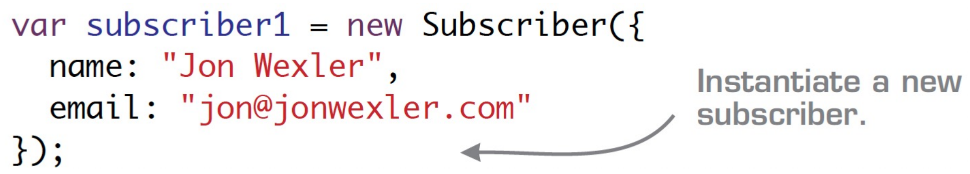
- Once the schema is defined, apply it to a model:

```
const Subscriber = mongoose.model("Subscriber", subscriberSchema).
```

- The model method takes a model name of your choice and a previously defined schema
  - in this case, the subscriberSchema
- The model is what you'll use to instantiate new Subscriber objects, and the schema you created can be used for that model.

Instantiating New Objects

- You can instantiate new objects from the model by referring to Subscriber.
  - construct a new instance of the Subscriber model

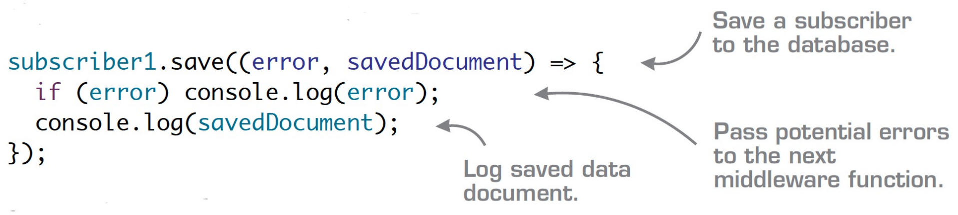


```
var subscriber1 = new Subscriber({
  name: "Jon Wexler",
  email: "jon@jonwexler.com"
});
```

Instantiate a new subscriber.

Storing New Objects in Database

- To save the newly created Subscriber object in the database, call save method on it
  - then handle any errors or returned data through a callback function.



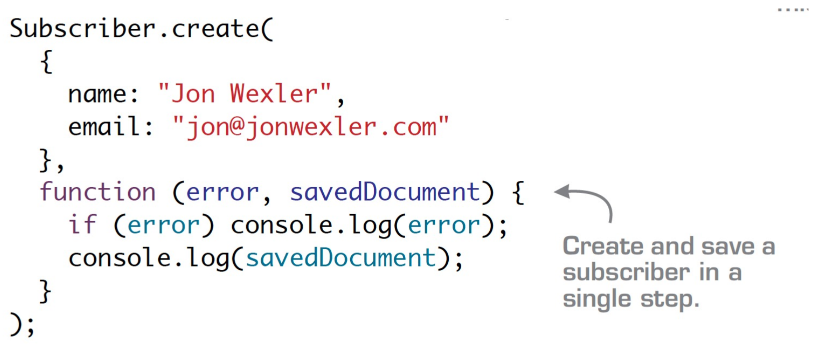
```
subscriber1.save((error, savedDocument) => {
  if (error) console.log(error);
  console.log(savedDocument);
});
```

Save a subscriber to the database.
Pass potential errors to the next middleware function.
Log saved data document.

  - an error may happen when data doesn’t match the schema types
  - the saved item returns data that you can use elsewhere in the application.

Instantiate and Save New Objects

- create() method does what new keyword and save() method do in one step.



```
Subscriber.create(
  {
    name: "Jon Wexler",
    email: "jon@jonwexler.com"
  },
  function (error, savedDocument) {
    if (error) console.log(error);
    console.log(savedDocument);
  }
);
```

Organizing Your Models

- Organize your models so that they don't clutter your main file.
- As you do for your views and controllers, create a models folder at the root level of your application.
  - create a new file called subscriber.js within the folder
  - this file is where you'll move your model's code.
  - move all the schema and model definition code to this file
  - add the model to the file's exports object.

```
const mongoose = require("mongoose"),
  subscriberSchema = mongoose.Schema({
    name: String,
    email: String,
    zipCode: Number
  });

module.exports = mongoose.model("Subscriber", subscriberSchema);
```

Export the Subscriber model as the only module export.

Organizing Your Models

- require the model in your main file:

```
const Subscriber = require("./models/subscriber")
```

- Now you should be able to use the model the same way as before.

Working Around with Mongoose

- To find documents in your database, use Mongoose’s findOne and where query methods.

```
Subscriber.findOne({ name: "Jon Wexler" }).where("email", /wexler/)
```

- Mongoose lets you chain parts of a query and even store queries in a variable.

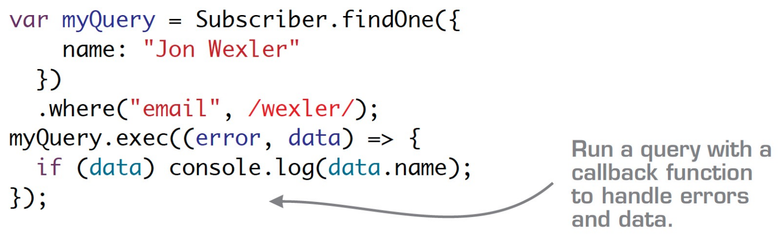
```
var findWexlers = Subscriber.findOne({ name: "Jon Wexler" }).where("email", /wexler/);
```

- Then you could run the query later by:

```
findWexlers.exec()
```

Working Around with Mongoose

- If you plan to run a query with the exec method, you need a callback function with two arguments:



```
var myQuery = Subscriber.findOne({
    name: "Jon Wexler"
  })
  .where("email", /wexler/);
myQuery.exec((error, data) => {
  if (data) console.log(data.name);
});
```

Run a query with a callback function to handle errors and data.

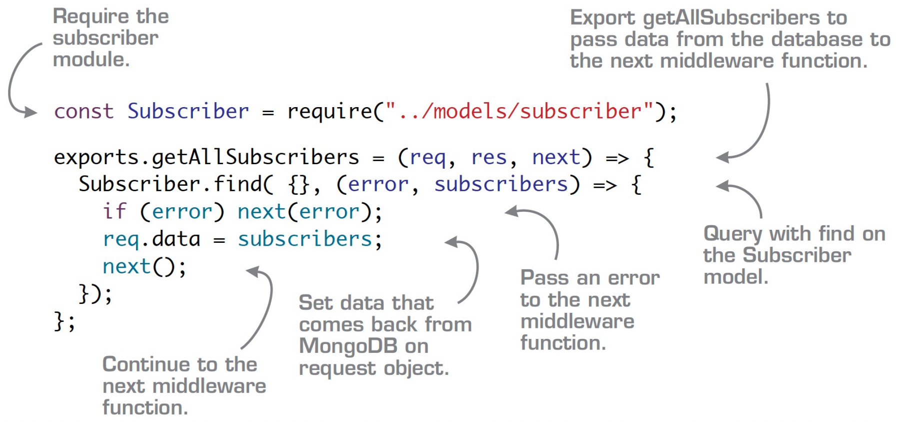
  - the first argument represents any errors that occur
  - the second argument represents any data returned by the database

Creating a Controller for Subscribers

- Having a model set up, create a controller that handles external requests specifically looking for data related to your model.
- Now that someone may request to register as a subscriber, you need to implement a subscriber controller:
  - Create a new file in your controllers folder called subscribersController.js
  - The file needs access to mongoose and your Subscriber model, require them.

Creating a Controller for Subscribers

- Create a controller action for when a request is made to view all subscribers in your database.



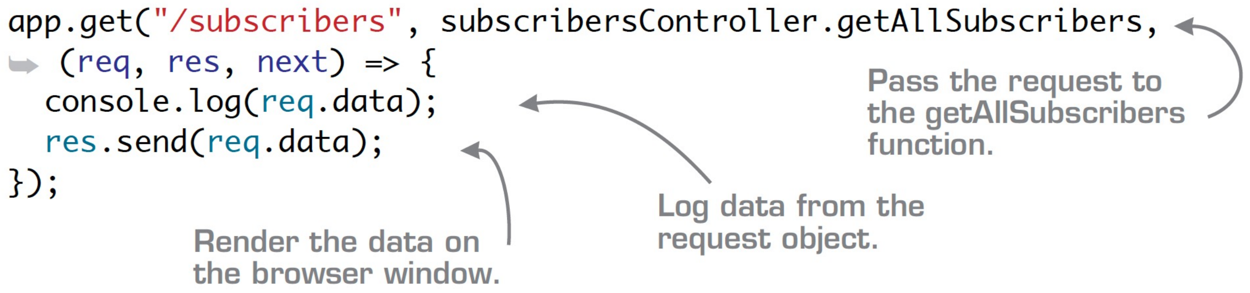
```
const Subscriber = require("../models/subscriber");

exports.getAllSubscribers = (req, res, next) => {
  Subscriber.find( {}, (error, subscribers) => {
    if (error) next(error);
    req.data = subscribers;
    next();
  });
};
```

Require the subscriber module.
Export getAllSubscribers to pass data from the database to the next middleware function.
Query with find on the Subscriber model.
Pass an error to the next middleware function.
Set data that comes back from MongoDB on request object.
Continue to the next middleware function.

Setting up a Route

- The next step is setting up the route in main file.



```
app.get("/subscribers", subscribersController.getAllSubscribers,
➥  (req, res, next) => {
  console.log(req.data);
  res.send(req.data);
});
```

Pass the request to the getAllSubscribers function.
Log data from the request object.
Render the data on the browser window.

  - Make sure to require the subscribers controller:

```
const subscribersController=require("./controllers/subscribersController")
```

Render Data on a View

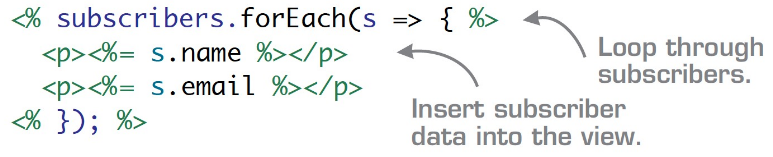
- Modify the action’s return statements to respond with rendering a view

```
res.render("subscribers", {subscribers: req.data})
```

  - The response makes a call to render a view called subscribers.ejs
  - It passes the subscribers from the database to that view via subscribers.
  - Build the view to display these subscribers.

Creating the View

- Create a file in views folder called subscribers.ejs:



```
<% subscribers.forEach(s => { %>
  <p><%= s.name %></p>
  <p><%= s.email %></p>
<% }); %>
```

Loop through subscribers.
Insert subscriber data into the view.

- Your view at http:// localhost:3000/subscribers should list your subscribers

Thank You!