The MEAN Stack

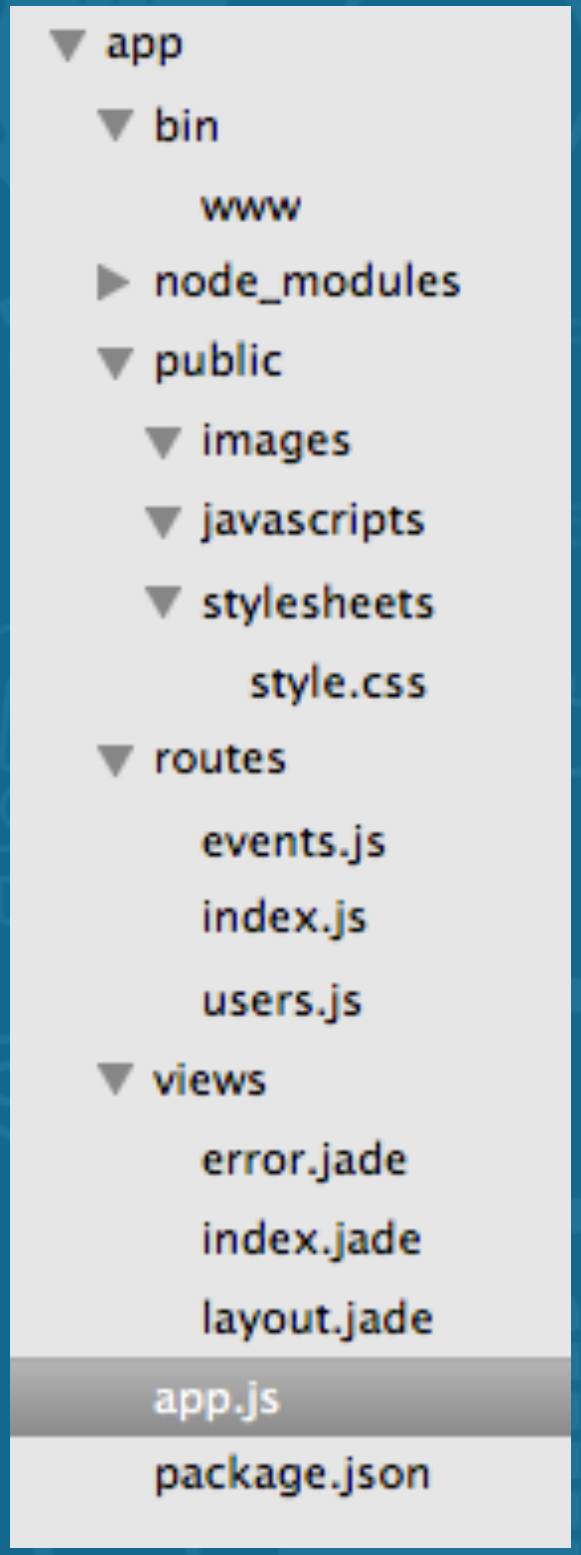
Week 2 - Express & MongoDB

Recap

- Created our skeleton Express server application
- Created API Route: **localhost:3000/events**

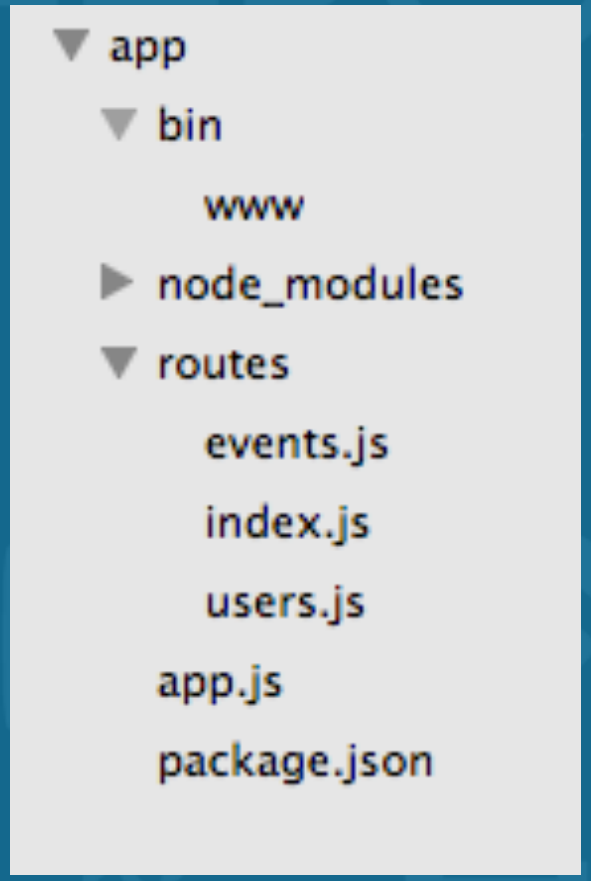
House Keeping

- Clean up our directory



```
▼ app
  ▼ bin
      www
  ▶ node_modules
  ▼ public
    ▼ images
    ▼ javascripts
    ▼ stylesheets
        style.css
  ▼ routes
      events.js
      index.js
      users.js
  ▼ views
      error.jade
      index.jade
      layout.jade
    app.js
    package.json
```

Before



```
▼ app
  ▼ bin
      www
  ▶ node_modules
  ▼ routes
      events.js
      index.js
      users.js
    app.js
    package.json
```

After

House Keeping

- Remove template and view rendering from **app.js**

```
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');
app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(express.static(path.join(__dirname, 'public')));
```

House Keeping

- Move error handling from **app.js** into a new file called **errors.js** (in project root directory)

```
module.exports = function (app) {

  app.use(function(req, res, next) {
    var err = new Error('Not Found');
    err.status = 404;
    next(err);
  });

  if (app.get('env') === 'development') {
    app.use(function(err, req, res, next) {
      res.status(err.status || 500);
      res.send({
        message: err.message,
        error: err
      });
    });
  }

  app.use(function(err, req, res, next) {
    res.status(err.status || 500);
    res.send({
      message: err.message,
      error: {}
    });
  });
}
```

House Keeping

- Our **app.js** should now look like this

```
var express = require('express');
var path = require('path');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var app = express();

app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());

var routes = require('./routes/index');
var users = require('./routes/users');
var events = require('./routes/events.js');

app.use('/', routes);
app.use('/users', users);
app.use('/events', events);

var errorHandler = require('./errors.js')(app);

module.exports = app;
```

Helpers

- Install *nodemon* to restart our server when files change
  - **$ npm install -g nodemon**
- Create a file that starts our server
  - File: **start.js** (in project root directory)
  - First Line: **require("./bin/www");**
- Now we start our server like this
  - **$ nodemon start.js**

Helpers

- Postman Chrome Extension (www.getpostman.com)
- Make HTTP requests from your browser
- Great for testing your API endpoints

Today

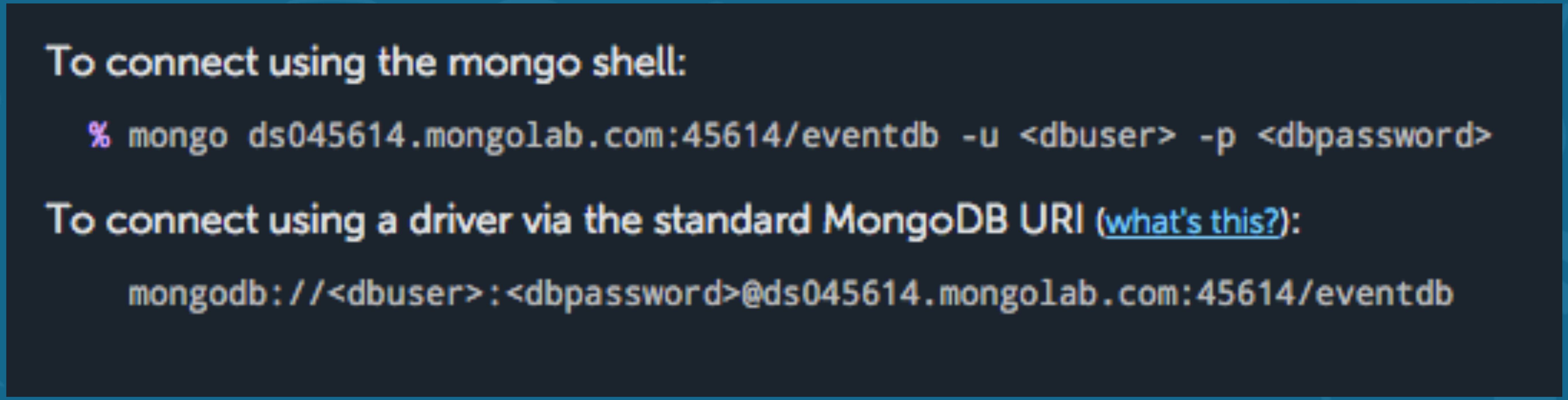
- Setup MongoDB
- Connect to MongoDB through Mongoose
- Create data model for Events
- Define API endpoints for Events
- Implement API endpoints (lets see how far we get)

MongoDB

- MongoLab (https://mongolab.com/) Database-As-A-Service
- Register, confirm verification email, log in, and click ⚡ Create new
- Use the following settings in “Create new subscription” form
  - Cloud Provider: Amazon Web Services
  - Plan: Single Node - Standard Line - Sandbox
  - Database Name: **eventdb**
- Click ⚡ Create new MongoDB deployment to launch your database

MongoDB

- Select your database from “MongoDB Deployments”
- Database Connection Details:

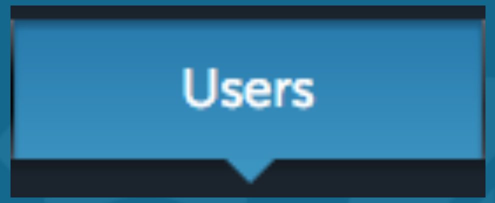
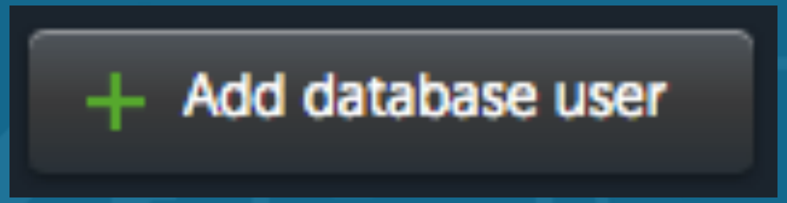


To connect using the mongo shell:

```
% mongo ds045614.mongolab.com:45614/eventdb -u <dbuser> -p <dbpassword>
```

To connect using a driver via the standard MongoDB URI (what's this?):

```
mongodb://<dbuser>:<dbpassword>@ds045614.mongolab.com:45614/eventdb
```

- Let’s create a User to access the database
  - Click , then click 
  - For simplicity, use your same Username / PW

Mongoose

- Mongoose will allow us to connect and query our Mongo database
- Lets install from NPM (make sure we're in our project root)
  - **$ npm install mongoose --save (two dashes)**
  - The --save argument will add the Mongoose dependency to our **package.json** file

Mongoose

- Create a connection to our database from **app.js**

```
var mongoose = require("mongoose");
mongoose.connect("mongodb://<dbuser>:<dbpassword>@ds045614.mongolab.com:45614/eventdb");

// app.use() declarations..
```

- Create our event data model in **/models/event.js**

```
var mongoose = require("mongoose");
var Schema = mongoose.Schema;

var EventSchema = new Schema({
  name: String,
  date: Date,
  description: String
});

module.exports = mongoose.model('Event', EventSchema);
```

API Endpoints

- Define how mobile, and web apps will interface with our application
    - Events must be CRUD (Created - Read - Updated - Deleted)
    - We must define HTTP endpoints to route this functionality
        - **Remember HTTP Verbs: PUT, GET, POST, DELETE**
        - We'll map the verbs to the respective CRUD functions

API Endpoints

- Last week, we created endpoint and Router in **/routes/events.js**

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.send('events!');
});

module.exports = router;
```

- Loaded the Event router in **app.js**

```
var routes = require('./routes/index');
var users = require('./routes/users');
var events = require('./routes/events.js');

app.use('/', routes);
app.use('/users', users);
app.use('/events', events);
```

Express Routing

- **app.use(path, function)**
- Express will send all requests matching the path, to the router

```
var routes = require('./routes/index');
var users = require('./routes/users');
var events = require('./routes/events.js');

app.use('/', routes);
app.use('/users', users);
app.use('/events', events);
```

- ex. **app.use('/apple', ...)** will match
  - “/apple”
  - “/apple/images”
  - “/apple/images/news”, …

Endpoint Definition

- An HTTP **GET** API endpoint is defined for the route “/”
- Endpoints have a function to handle request, as such they have request, and response objects as arguments

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.send('events!');
});

module.exports = router;
```

- **router.HTTP_VERB(path, function)**
- Export Router though **module.exports**

Event API Endpoints

Function	CRUD	HTTP Verb	Endpoint
Create an event	Create	POST	/events
Get all events	Read	GET	/events
Get an event	Read	GET	/events/:event_id
Update an event	Update	PUT	/events/:event_id
Delete an event	Delete	DELETE	/events/:event_id

Create Event

- API Endpoint: **POST localhost:3000/events**
- Get Event data from body of the Request
- Create new Event instance
- Save instance
- Send instance in Response

```
router.post('/', function (req, res, next) {

  var newEvent = new Event();

  newEvent.name = req.body.name;
  newEvent.date = req.body.date;
  newEvent.description = req.body.description;

  newEvent.save(function (err, savedEvent) {
    if (err) return next(err);
    return res.send(savedEvent);
  })

})
```

Get All Events

- API Endpoint: **GET localhost:3000/events**
- Query MongoDB Event Model for all Events
- We check for errors
- Send query results back to client in the Response

```
router.get('/', function (req, res, next) {

  Event.find(function (err, events) {
    if (err) return next(err);
    return res.send(events);
  })

})
```

Get An Event

- API Endpoint: **GET localhost:3000/events/:event_id**
- The Event ID is passed in as a URL parameter
- Query for Event by ID
- Check for any errors
- Send query results back to client in the Response

```
router.get('/:event_id', function (req, res, next) {

  var event_id = req.params.event_id;

  Event.findById(event_id, function (err, event) {
    if (err) return next(err);
    return res.send(event);
  })

})
```

Update An Event

- API Endpoint: **PUT localhost:3000/events/:event_id**
- The Event ID is passed in as a URL parameter
- Query for Event by ID
- Update attributes and save
- Send query results back to client in the Response

```
router.put('/:event_id', function (req, res, next) {
  var event_id = req.params.event_id;
  Event.findById(event_id, function (err, event) {
    if (err) return next(err);

    if (req.body.name) event.name = req.body.name;

    if (req.body.date) event.date = req.body.date;

    if (req.body.description) {
      event.description = req.body.description;
    }

    event.save(function (err, savedEvent) {
      if (err) return next(err);
      return res.send(savedEvent);
    })

  })

})
```

Delete An Event

- API Endpoint: **DELETE localhost:3000/events/:event_id**
- The Event ID is passed in as a URL parameter
- Query and Delete Event using Event ID
- Send client a success

```
router.delete('/:event_id', function (req, res, next) {

  var event_id = req.params.event_id;

  Event.findByIdAndRemove(event_id, function (err) {
    if (err) return next(err);
    res.send({message: "Event Deleted"});
  })

})
```

Done

- Next Week
  - Search for Events
  - Define User Model
  - Implement User Authentication