



1) Log into github and create a new repository

GitHub, Inc. [US] | <https://github.com/orgs/EmpowerCourse/dashboard>

Search or jump to... Pull requests Issues

< EmpowerCourse

Repositories **New repository**

Find a repository...

EmpowerCourse/course_content

EmpowerCourse/classmates

Browse activity

tahbaza pushed to Er

1 commit to mast

886582e added a

tahbaza pushed to Er

1 commit to mast

6fe9ff9 added d

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

<E EmpowerCourse ▾

Repository name

/ api-voting ✓

Great repository names are short and memorable. Need inspiration? How about **reimagined-train**.

Description (optional)

This is a simple node and mongo app to accept user votes

☒ Public

Anyone can see this repository. You choose who can commit.

☐ Private

You choose who can see and commit to this repository.

☐ Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None ▾

Add a license: None ▾



Create repository

Copy the github repo url you just created:

EmpowerCourse / api-voting

Unwatch ▾ 2

★ Star 0

🍴 Fork 0

<> Code

🔔 Issues 0

🔗 Pull requests 0

📁 Projects 0

📖 Wiki

📊 Insights

⚙️ Settings

Give access to the people you work with

You should give access to the collaborators and teams you need to work with.

Add teams and collaborators

Quick setup — if you've done this kind of thing before

📥 Set up in Desktop

 or

HTTPS

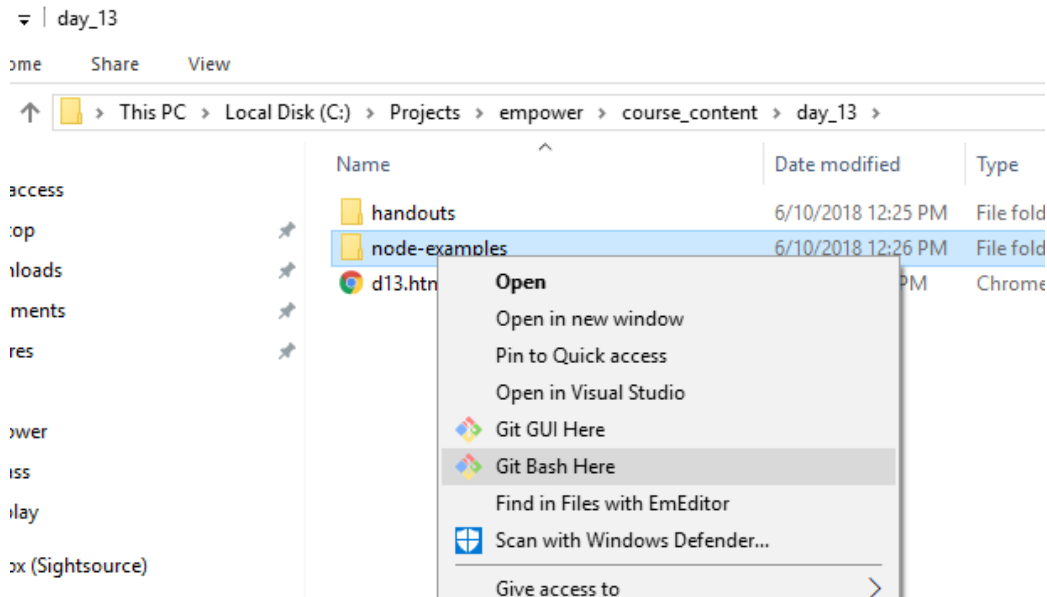
SSH

git@github.com:EmpowerCourse/api-voting.git

📄

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

- 2) Navigate to the directory that will contain your new project directory and open a new git bash window there.



- 3) express is a very simple node web server that will enable us to respond to our simple requests appropriate. Install it by running the following command which tells node package manager (npm) to download and install express-generator “globally” (that’s the `-g`).

npm install express-generator -g

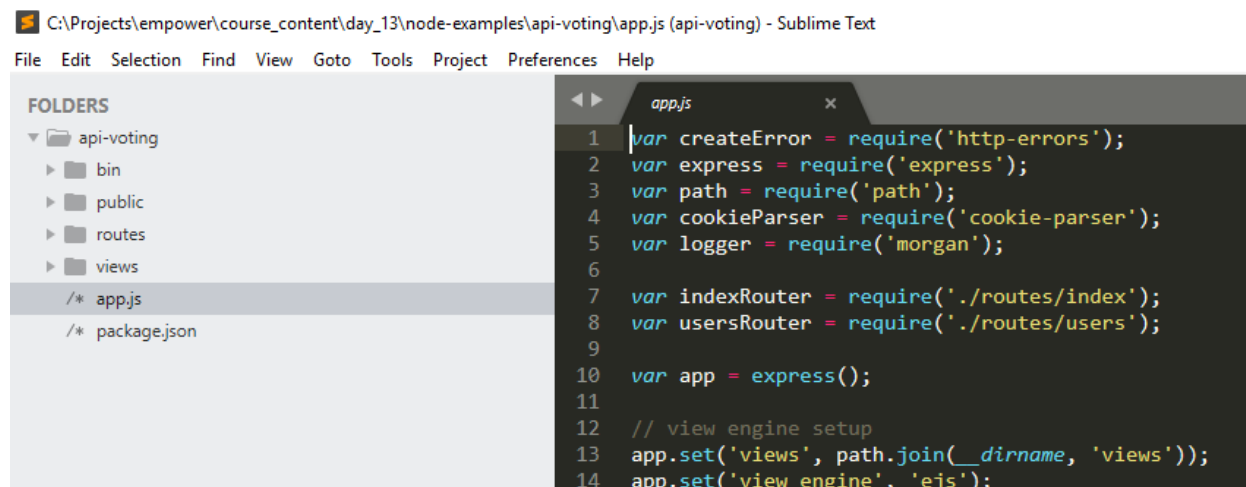
- 4) Lets use the express generator we just installed to create a skeleton project with a single command

express --view=ejs api-voting

- 5) That created a subdirectory that contains our skeleton project. Let’s navigate into the new directory.

cd api-voting

- 6) Open the directory we just created in your favorite development text editor (probably with a file/open folder menu click).



- 7) Type **npm init** in order to create a new node project. It will prompt you for a bunch of answers so that it can go off and generate files and directories for you to save time. Here are appropriate answers for this project. Type your name instead of mine obviously and your git repo url rather than mine.

```

$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit
package name: (examples) api-voting
version: (1.0.0)
description: This is a simple node and mongo app to accept user votes
entry point: (index.js)
test command:
git repository: git@github.com:EmpowerCourse/api-voting.git
keywords:
author: Ali Tahbaz
license: (ISC)
About to write to C:\Projects\empower\course_content\day_13\node-examples\package.json:

{
  "name": "api-voting",
  "version": "1.0.0",
  "description": "This is a simple node and mongo app to accept user votes",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+ssh://git@github.com:EmpowerCourse/api-voting.git"
  },
  "author": "Ali Tahbaz",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/EmpowerCourse/api-voting/issues"
  },
  "homepage": "https://github.com/EmpowerCourse/api-voting#readme"
}

Is this ok? (yes) yes

```

- 8) Now let's install some other node packages (components that add features to our application). The package.json file is an inventory of all dependencies/components that your application needs to run. The `--save` parameter in a flag/parameter that tells npm to not only install the package for your application but also add the dependency to your package.json file as a dependency to remember. This is important because this list of dependencies will be used to install those bits of code on other computers, most importantly the eventual Internet server you use for deployment to share with others.

The body-parser will enable your node web server application to parse FORM data into data that can be used by your web server actions.

The mongoose package is a component that is designed to make it easy for us to send and receive data to a MongoDB non-relational database.

```


npm install body-parser --save
npm install mongoose --save

```

- 9) Let's go create a database to put our data in. Go to <https://www.mongodb.com/cloud> and create a free account by clicking the Get Started Free button in the middle of the screen.

Get started free

- 10) Enter your desired account details.

 Account Profile

Email Address

s2empower@gmail.com

Password

.....

✓ 8-characters-minimum ✓ One-number
✓ One-letter ✓ One-special-character

First Name Last Name

S2 Empower

Phone Number Company Name

3367213287 S2

Job Function

Software Developer / Engineer ▼

Country

United States ▼

☒ I agree to the [terms of service](#)

Already have an account? [Login](#) [Continue](#)

- 11) Click the Continue button and you'll be presented with a bunch of cluster config options. We're going to accept the defaults on the first screen by clicking the following button.

NEXT: CLUSTER TIER

- 12) The default "tier" in the next panel will be FREE, what we want, so accept the default and click this button.

NEXT: ADDITIONAL SETTINGS

- 13) Accept the defaults on this panel too and click this button.

NEXT: CLUSTER NAME

- 14) Finally, be sure to enter your cluster name and then click the Create Cluster button.

Cluster Name

empower

One time only: once your cluster is created, you won't be able to change its name.

empower

Cluster names can only contain ASCII letters, numbers, and hyphens.

PREVIOUS: ADDITIONAL SETTINGS

FREE

Pay-as-you-go! You will be billed hourly and can terminate your cluster anytime. Excludes variable *data transfer*, *backup*, and taxes.

Cancel

Create Cluster

15) Click the Security tab and add a user account that you'll use to connect.

S2 > PROJECT 0

Clusters

Overview

Security

MongoDB Users

IP White

User Name

+ ADD NEW USER

Choose a good username and click the Autogenerate Secure Password to generate a good, secure password. Leave the permissions for this user as Read and write to any database and then click the Add User button. **Copy/paste and save this password somewhere** as we're going to need it in a minute.

Add New User

SCRAM Authentication

SCRAM is MongoDB's default authentication method.

db-user

e.g. new-user_31

B3gONvi32J81FH

Autogenerate Secure Password

User Privileges

Atlas admin

Read and write to any database

Only read any database

Show Advanced Options

Cancel

Add User

- 16) Lets get our database connection details. Click back over on the Overview tab and then click the Connect button. Click the Add Current IP Address button and then the Save button that is displayed. Remember where this whitelist button is. We will also need to add our Internet server's ip address here when we deploy to the Internet.

Clusters

S2 > PROJECT 0

Clusters

Overview Security

Find a cluster...

SANDBOX

empower

Version 3.6.5

METRICS CONNECT

INSTANCE SIZE

M0

REGION

AWS / N. Virginia (us-east-1)

TYPE

Replica Set - 3 nodes

LINKED STITCH APP

None Linked - Link Application

Operations

Last 6 Hours

Connections

Last 6 Hours

Connect to empower

Your IP whitelist is empty. You can only connect to your cluster from whitelisted IP addresses. You can add to your whitelist below.

1 Check the IP Whitelist

You don't have any IP addresses in your whitelist. You won't be able to connect without one.

+ ADD ENTRY

ADD CURRENT IP ADDRESS

ALLOW ACCESS FROM ANYWHERE

2 Choose a connection method:

Connect with the Mongo Shell

Mongo Shell with TLS/SSL support is required

Connect Your Application

Get a connection string and view driver connection examples

Connect with MongoDB Compass

Download Compass to explore, visualize, and manipulate your data

See methods to add data and diagnostics in the [Command Line Tools](#) shortcut from within your cluster.

Close

- 17) Click the Connect to your Application button and then the “I am using driver 3.4 or earlier” button to display the connection details we’ll need from our node app. Click the COPY button. Paste into a text editor and replace the <PASSWORD> bit with your real database user’s password. We’re going to use this in a minute by giving it to our node application so it knows where its database is.

1 Copy a connection string:

[See documentation on how to check the version of your driver](#)

I am using driver 3.6 or later

I am using driver 3.4 or earlier

Copy the URI connection string:

```
mongodb://db-user:<PASSWORD>@empower-shard-00-00-  
nuc23.mongodb.net:27017,empower-shard-00-01-  
nuc23.mongodb.net:27017,empower-shard-00-02-  
nuc23.mongodb.net:27017/test?ssl=true&replicaSet=empower-shard-  
00&authSource=admin&retryWrites=true
```

COPY

- 18) Let’s create the collections we’ll need to hold our data. To do so we’ll need to download and use an application whose purpose in life is to enable developers to manage their MongoDB data. Click the Connect button again on your cluster and then the Connect with MongoDB Compass button and then the Operating System button to download and install the application.

Connect with MongoDB Compass

Download Compass to explore, visualize, and manipulate your data



- 19) While it’s downloading click the COPY button to capture the connection details you’ll need.

Connect to Compass

1 If you have not already, click below to download Compass

Windows

Mac OS X

Other Operating Systems ▾

2 Copy the URI Connection String

[View detailed instructions](#)

I am using Compass 1.12 or later

I am using Compass 1.11 or earlier

```
mongodb+srv://db-user:<PASSWORD>@empower-nuc23.mongodb.net/admin
```

COPY

Replace **PASSWORD** with the password for the *db-user* user.

When you open Compass, it should detect the URI string from your clipboard and auto-populate the form.

20) When the MongoDB Compass application loads it should detect the value in the connection string and put the necessary values in the required boxes for you. You'll just need to enter the password for the database user. Click Create Favorite, Save Favorite and then Connect.

Hostname

empower-nuc23.mongodb.net

SRV Record

☒

Authentication

Username / Password ▼

Username

db-user

Password

Authentication Database ⓘ

admin

Replica Set Name

Read Preference

Primary ▼

SSL

System CA / Atlas Deployment ▼

SSH Tunnel

None ▼

Favorite Name ⓘ

EmpowerDB

CREATE FAVORITE

CONNECT

21) Now let's create a database with the collections we'll need for our simple app. Mongo needs an initial collection to be created with the database container so we're going to create our database named api_votes and our first collection named api here:

Databases

CREATE DATABASE

Database Name

api_votes

Collection Name

api

☐ Capped Collection ⓘ

Before MongoDB can save your new database, a collection name must also be specified at the time of creation. [More Information](#)

CANCEL **CREATE DATABASE**

22) Now let's create 2 more collections named voter and vote. To do so click the database name...

Databases

CREATE DATABASE

Database Name

admin

api_votes

local

...and then click the Create Collection button.

CREATE COLLECTION

Create Collection

Collection Name

voter

☐ Capped Collection ⓘ

CANCEL **CREATE COLLECTION**

23) We've downloaded and installed packages that we need for node and created a database with a couple of collections in MongoDB but we haven't told our application to make use of them.

Go back to your text editor showing you the node application directory. The **app.js** file located in your application root directory is what node will run to start up your application. This is most often where application setup and configuration occurs in node.

Let's modify the default lines in the file to include our new body-parser and mongoose components by adding or modifying the highlighted lines here. The red, highlighted text below is from **step 17** above.

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser'); // reference our body parser component
var logger = require('morgan');
// this is the start of our MongoDB database connectivity, the next lines tell
// our app where our database is, our security credentials and generally
// how to speak with the database
var mongoose = require('mongoose');
// paste this path from your MongoDB cloud account info panel
var mongoDBPath = 'mongodb://some-user:some-user-password@empower18-ws1-shard-00-00-pgyeb.mongodb.net:27017,empower18-ws1-shard-00-01-pgyeb.mongodb.net:27017,empower18-ws1-shard-00-02-pgyeb.mongodb.net:27017/api_votes?ssl=true&replicaSet=empower18-ws1-shard-00&authSource=admin&retryWrites=true';
dbOptions = {
  keepAlive: 1000,
  connectTimeoutMS: 30000,
  native_parser: true,
  auto_reconnect: false,
  poolSize: 10
};
mongoose.connect(mongoDBPath, dbOptions);
mongoose.Promise = global.Promise;
const db = mongoose.connection;
db.on('error', (err) => {
  console.log(err);
});
db.once('open', (err) => {
  if (err) {
    console.log(err);
  }
});

// this file contains the default page that will be sent back upon a request
var indexRouter = require('./routes/index');
// this was just an example route, we can comment it out
// var usersRouter = require('./routes/users');
var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(logger('dev'));
```

```
// we're going to comment out the default express processor and
// tell it to use our slightly better body parser instead
// app.use(express.json());
// app.use(express.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
// comment out this path, no longer relevant
// app.use('/users', usersRouter);
// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});
// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});
module.exports = app;
```

24) Lets go add our application specific functionality. We need to create object definitions to mirror each of our record types. These are called “models”. Add a directory under your application root called models and add a file named **voter.js** within it that contains the following code.

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var Voter = new Schema({
  first_name : String,
  last_name  : String,
  username   : String
}, { collection: 'voter' });
module.exports = mongoose.model('Voter', Voter);
```

25) Create a file named **api.js** within that same models directory that contains the following code.

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var Api = new Schema({
  name : String
}, { collection: 'api' });
module.exports = mongoose.model('Api', Api);
```

26) Now create a file named **vote.js** within that same models directory that contains the following code.

```

var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var Vote = new Schema({
  voter_id: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Voter'
  },
  api_id: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Api'
  }
}, { collection: 'vote' });
module.exports = mongoose.model('Vote', Vote);

```

27) Let's add some basic logic to make the application features meet our needs. Open routes/index.js and add/modify to the following. We will discuss this in class in detail.

```

var express = require('express');
var router = express.Router();
var Voter = require('../models/voter');
var Vote = require('../models/vote');
var Api = require('../models/api');

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

function getApiById(api_list, api_id) {
  for(var i=0; i<api_list.length; i++) {
    if (api_id.equals(api_list[i].api_id)) {
      return api_list[i];
    }
  }
  return null;
}

// GET vote tally
router.get('/tally', function(req, res, next) {
  Vote.find().exec(function(err, votes) {
    if (err) return next(err);
    // first we group the results into a distinct list, counting votes for each api
    var prettyResults = [];
    for(var i=0; i<votes.length; i++) {
      var witnessedApi = getApiById(prettyResults, votes[i].api_id);
      if (witnessedApi === null) {
        prettyResults.push({api_id: votes[i].api_id, vote_count: 1});
      } else {
        witnessedApi.vote_count++;
      }
    }
    // now lets loop through the distinct list and add the names of the apis to help it make
    sense to the end user

```

```

    var promiseArray = prettyResults.map(tally_row => new Promise(function(resolve, reject) {
        Api.findById(tally_row.api_id, function (err, api) {
            if (err) return reject(err);
            tally_row.api_name = api.name;
            resolve(tally_row);
        });
    }));
    Promise.all(promiseArray).then(function(results) {
        res.json(results);
    });
});

// GET vote tally
router.get('/apis', function(req, res, next) {
    Api.find().exec(function(err, apis) {
        if (err) return next(err);
        res.json(apis);
    });
});

// GET user's vote
router.get('/:username/vote', function(req, res, next) {
    // find the referenced voter
    Voter.find({'username': req.params.username}).exec(function(err, voter) {
        if (err) return next(err);
        if (voter.length == 0) {
            return res.json({
                success: false,
                message: "No voter with username " + req.params.username + " could be found!"
            });
        }
        // find the first vote recorded for this voter (should only be 1)
        Vote.find({'voter_id': voter[0]._id}).exec(function(err, existing_votes) {
            if (err) return next(err);
            if (existing_votes.length == 0) {
                return res.json({success: false, message: req.params.username + " has not yet
voted."});
            }
            // go get the api record related to this vote
            Api.findById(existing_votes[0].api_id, function (err, api) {
                if (err) return next(err);
                // return the vote information
                return res.json({
                    success: true,
                    message: {
                        api_name: api.name,
                        voter_username: voter[0].username
                    }
                });
            });
        });
    });
});

```

```

// POST to save a user's vote
// expects a json request specifying { api_name: "some-api" }
router.post('/:username/vote', function(req, res, next) {
  Api.find({'name': req.body.api_name}).exec(function(err, api) {
    if (err) return next(err);
    if (api.length == 0) {
      return res.json({success: false, message: "No api named " + req.body.api_name + "
could be found!"});
    }
    Voter.find({'username': req.params.username}).exec(function(err, voter) {
      if (err) return next(err);
      if (voter.length == 0) {
        return res.json({
          success: false,
          message: "No voter with username " + req.params.username + " could be found!"
        });
      }
      Vote.find({'voter_id': voter[0]._id, 'api_id': api[0]._id}).exec(function(err,
existing_votes) {
        if (err) return next(err);
        // only 1 vote per voter is allowed
        if (existing_votes.length > 0) {
          return res.json({
            success: false,
            message: req.params.username + " has already voted once! Delete the vote in
order to vote again if you like."
          });
        }
        // record a new vote for this voter
        var vote = {
          voter_id: voter[0]._id,
          api_id: api[0]._id
        };
        new Vote(vote).save(function (err) {
          if (err) return next(err);
          console.log('inserted');
          res.json({success: true, message: "Vote recorded"});
        });
      });
    });
  });
});

// DELETE to remove a voter's vote
router.delete('/:username/vote', function(req, res, next) {
  Voter.find({'username': req.params.username}).exec(function(err, voter) {
    if (err) return next(err);
    if (voter.length == 0) {
      return res.json({
        success: false,
        message: "No voter with username " + req.params.username + " could be found!"
      });
    }
    Vote.find({'voter_id': voter[0]._id}).exec(function(err, existing_votes) {
      if (err) return next(err);

```

```

    if (existing_votes.length > 0) {
      Vote.findByIdAndRemove(existing_votes[0].id, (err, b) => {
        if (err) return next(err);
        console.log('removed ' + b._id);
        res.json({success: true, message: "Vote removed"});
      });
    } else {
      return res.json({
        success: false,
        message: "No existing vote exists to delete for "+ req.params.username + "."
      });
    }
  });
});
});

module.exports = router;

```

28) We now have a working node.js app! Lets install our missing dependencies and run it locally. Go back to your git bash window and execute the following 2 commands.

```

npm install
npm start

```

At this point you've just started up your web server software on your local computer and it will stay running until you stop it.

29) Now open a web browser and navigate to <http://localhost:3000/tally> and you should get an empty Js array back indicating no results have yet been stored. The git bash window will log every interaction with the web server software you just built and any console.log messages so that you can see what's going on.