Node.js NPM – JavaScript runtime and server side tech & npm command Line manager (used to install modules)

Express framework – backend framework

Mongo Db

Mongoose ORM object relational mapper – Used by application to interface with MongoDB

Pug Template Template Engine ( called JADE before )

Bootstrap(twriter) – Bower Package Manage

JQuery & Ajax ( make delete request to server )

Git Bash ( Terminal / Command Line )

Atom uses Electron ( javascript framework ) used to build desktop application w/ javascript

NodeJs manually handle request and response ( get req. send response) – Express Helps in this.

Apache and PHP – apche server constantly runs and handles all the requests then upload php file

Created JSON File – npm init

Npm install –save pug

No html tags in pug ( use space or tab & maintain same level to keep with html tag to put content within parent element )

htmlTag= integer ( = must be inserted between htmltag and integer output. EqualTo sign lets pug know that it’s a number output )

Doctype

Html

Head

Title

--------

res.render(“index”)   
or res.render(‘index’ , {   
**titles**: ‘hello’ }) - no quotations for **variable**

IN index.pug parse **variable** with #(**titles**) - no quotations for **variable**

Npm install –g nodemon ( to avoid restarting server every time you make changes to JS file )

Template = View

Create new view ‘/articles/whatever’

Self closing tags cannot contain nested content!

Add layout to each view

Doctype

Html

Head

Title

**Block content**

P This is the footer

Footer

In each view at the TOP

Extends layout

Block content

IN pug file

Connect database with application using mongoose

mongoose.connect('mongodb://localhost/nodekb'); 🡨 /Name of database

let db = mongoose.connection;

Mongo db is very unstructured. Mongoose gives us the flexibility to structure the data on the application level rather than doing it on the database level ( like MYSQL )

Create models folder – Then article.js

Require mongoose

Create article schema

Schema contains title: , author: and body: to define and match the information being received from the db.

Then in app.get function (of home route ‘/’ ) add code to grab information from the data base and display onto the page

// Check connection

db.once('open' , function (){

console.log("Connected to MongoDB")}); 🡨 Will display in terminal

// Check for DB errors

db.on('error' , function (err){

console.log("err")}); 🡨 Will display in terminal

console.log() <-- submits to the database log / server log not the client’s log window

create form (method='POST', action='/articles/add' )for input of title, author and body and submit button - on pug page using bootstrap classes

Add submit post route on main js page ( app.post () )

Two request can be defined to the same URL as long as it’s a different type of request ( app.get VS app.Post )

**BODY-Parser** – Required Middle Ware ( npm install - - save body-parser ) ( required middle-ware code ) ( require body-parser)

**Body-parser** – used to capture input from body to database

Set variables to capture info from body = req.body.title req.body.author etc

.save() if err log err else res.redirect (‘/’) to home page – SAVES into database

create public folder to server static files like anything.html

app.use(express.static(path.join(\_\_dirname, 'public')));

Bootstrap / Bower.js / Jquery(included)

Install bootstrap bower creates a folder with contents in a default directory in the root – inorder to create the folder in the required folder – to be loaded statically

Create file in root. Name it .bowerrc 🡪 set { “directory” : “ desired/path”}