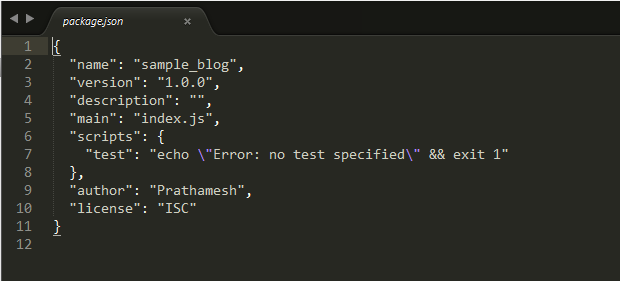
# **Sample BLOG App using MEAN Step-By-Step Guide**

# Initial Setup

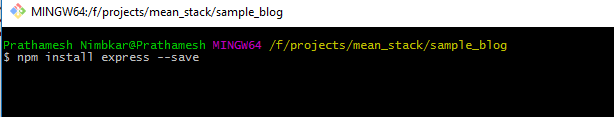
* Create project folder (say sample-blog)
* Route to that folder using command-prompt (Git bash)
* npm init : to create package.json



Package.json is as follows:



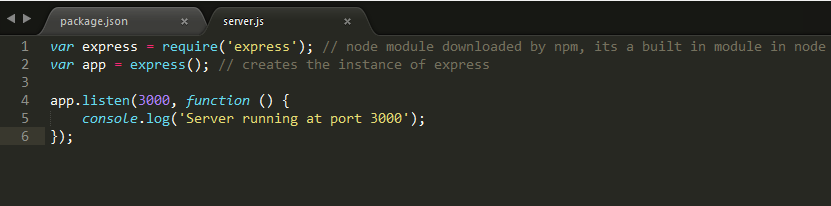
Install express: framework for nodejs via command npm install express –save



* - - save will save its dependencies in package.json and installs latest version of express.js to the application

# Create server.js file

* Create server.js file inside application folder with following code



# Create static files (inside public folder)

* Serve static content for the app from the “public” directory in the application directory
* Add following code in server.js



* Create folder name as ‘public’ inside application
* Add index.html with some dummy content as <h2>Hello World! MEAN - Sample Blog</h2>
* Run server.js , this would output as



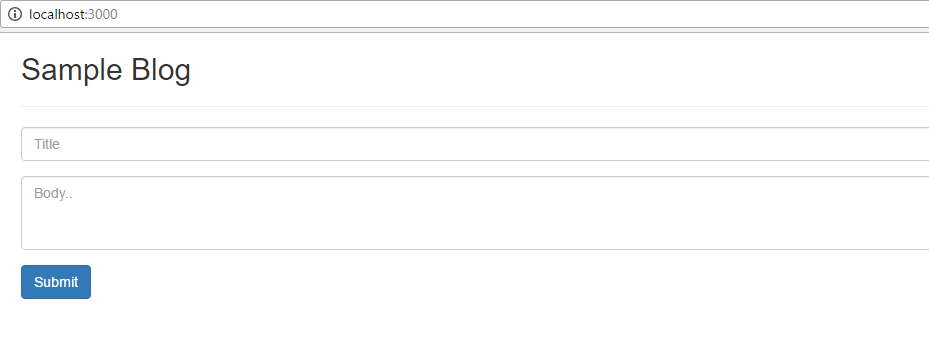
* Create <form> for entering Title and Blog using bootstrap
* Add AngularJS to libs folder and other related libraries in respective folder for eg : libs/css/bootstrap/bootstrap.css and so on…
* Add np-app=”SampleBlogApp” as module and add respective code in app/app.js



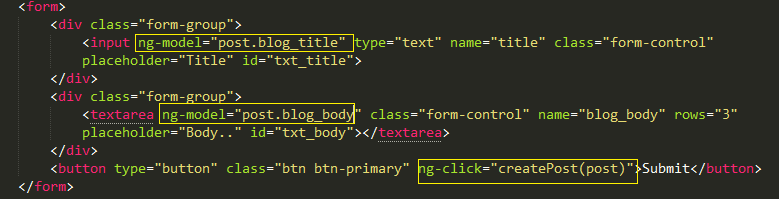
* The html/form be like



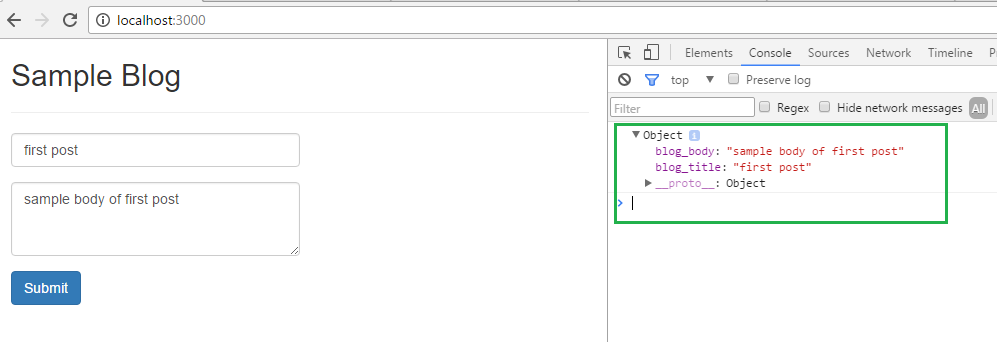
* This would output as



* Add code to get user inputs

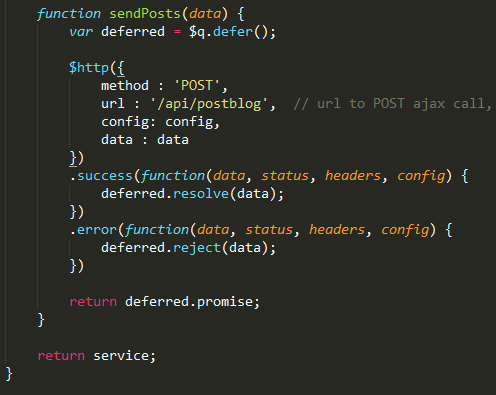


* This will output as



# POST form data to server

* Create Angular service as ‘SampleBlogService’ and inject $http to fire POST call to send the form data to server
* We use defer/promise



# Server side POST call handling

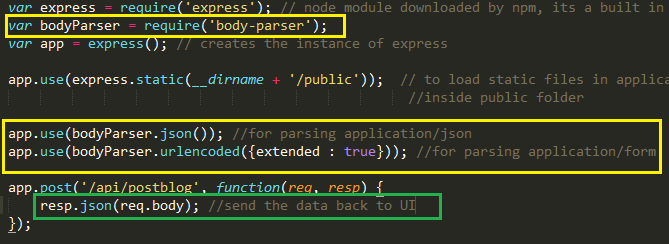
* Add POST call for handling REST call from UI side for url : /api/postblog
* With this call in place error 404 will be removed
* The server.js will have following call to handle this POST call



* This will output as “server responds to createPost ” on command lien console

# Add body-parser

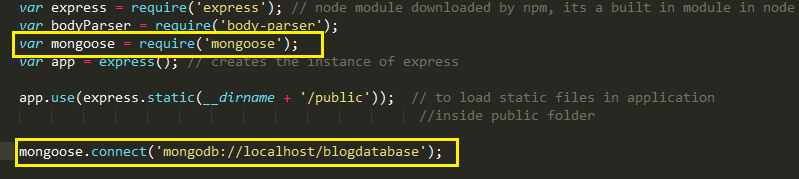
* Need to add body-parser module in application so as to read request payload by req.body
* Body-parser is module supported by node + express
* Parse incoming request bodies in a middleware before your handlers, availabe under the req.body property
* Body-parser can be imported by **npm install body-parser –save** this will load its dependencies in package.json

****

* **resp.json(req.body)** will send back the data to UI in JSON format with **Status: 200 OK**
* blog\_body:"sample blog body"
* blog\_title:"sample title"

# Create MongoDB database connection

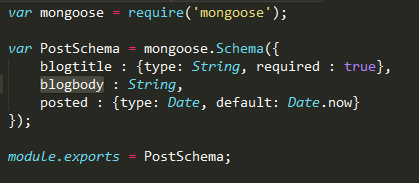
* Mongoose is a [MongoDB](https://www.mongodb.org/) object modeling tool designed to work in an asynchronous environment.
* Need to require mongoose in server.js file
* Before that need to import mongoose via **npm install mongoose –save** which loads all its dependencies in package.json
* Once we require mongoose inn server we need to create connection with mongoDB and this can be achieved by



* mongoose.connect('mongodb://localhost/blogdatabase'); will create connection with blogdatabse via localhost

# Create Schema

* create model folder under application
* create postmodel.js file inside model folder – this file contains database schema as follows

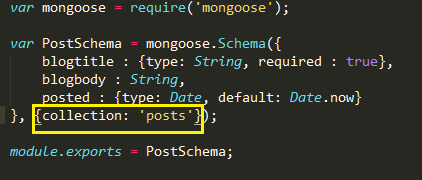


* blogtitle : {JS datatype as String, optional parameter as required}
* bolgbody: {JS datatype String}
* posted : {Date as JS datatype, and optional date formatter}
* this schema is imported as module in server.js file as

**var PostSchema = require('./model/postmodel');**

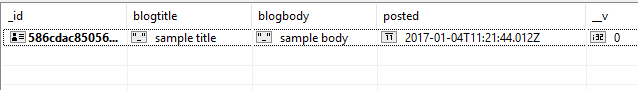
****

* above code creates collection as **postmodels** inside blogdatabase with the above schema
* **postmodels**: collection name created by mongoose as , it pluralize the model PostModel and makes it as postmodels
* this collection name can be changed to custom one as follows

****

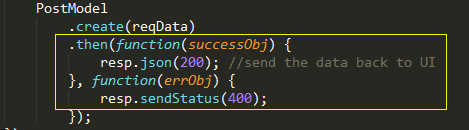
# View Database

* open mongo terminal/mongochef UI
* show databases
* will output as **blogdatabase**
* **use blogdatabase:** will take us inside this database
* **show collectons:** will show all collections i.**e. postmodels, posts**
* **db.posts.find()** will give us that form data has been successfully inserted in database

****

# Handle race condition

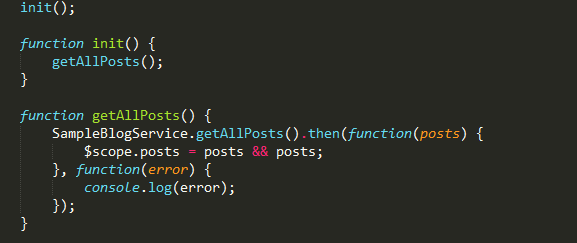
* need to handle race condition with help of promise object that database connection sends when the new record is added into the database or even if does not get added in database due to some errors , and this can be achieved as
* postmodel sends server side promise



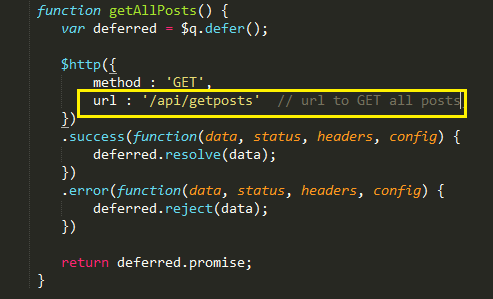
* sending the same object back to user what user has entered does not solve any purpose instead if it fails and user gets 404 as status then he would do the necessary action on UI

# GET Posts

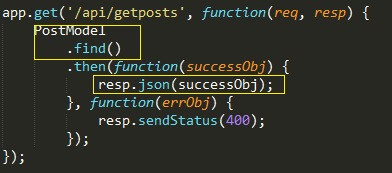
* loads all the posts from database to the UI
* inside controller create init() and call getAllPosts() from the init() to load all the previously available posts on UI
* app.js code will be as follows



* service call would be



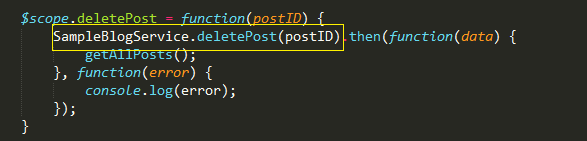
* url : /api/getposts is mapped to server.js GET call, which talks to database and gets data of all the available posts
* .find() called on PostModel which is nothing but db.posts.find() fired in mongodb to view all the databases
* On success callback server sends back the JSON output back to angular service from where its called to load the posts on GUI



* The same method can be called from success handler of createPost to view the newly created post along with pre-loaded posts from the database

# DELETE Post

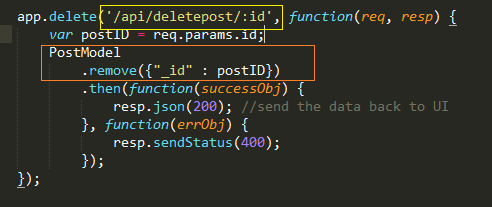
* To Delete post one-by-one we need to pass id which is unique to each and every post to server, so that server can then send this id to database and remove/delete post based on ID which user has sent
* **postID: post.\_id** sent by UI to ctrl which is nothing but unique ID which mongo has assigned to particular post in database



* angular service will then be called with delete method on $http object as follows
* url : '/api/deletepost/'+**postID :** postID will be ID of post which is to be deleted is passed in URL only



* server consumes this postId from url as **'/api/deletepost/:id'**
* postID = req.params.id will be unique id received from client
* **remove** method is called on this schema object PostModel with this postID/\_id as unique parameter

****

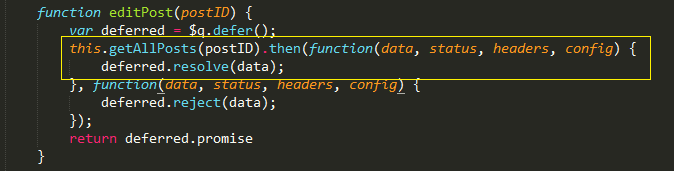
* this will then delete post in mongo db as per id and returns back the status in success handler
* success handler then sends back the status 200 OK to client side service
* if this happens to be succeeded, and after calling getAllPosts () will show the affected changes in UI with deletion of posts

# EDIT Post

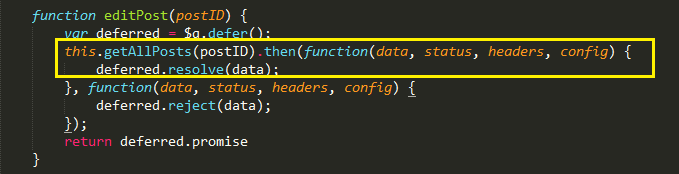
* To edit particular post we need to first get the post title and post body into the text boxes and then manually edit the reqd fields and send/update them back to server
* Need to add edit button for every post



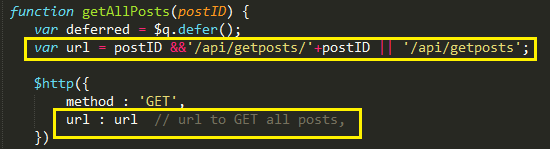
* Controller then sends postID of the one which is to be edited to get the details of the same into the text-boxes
* AngularJS service gets the data for requested postID



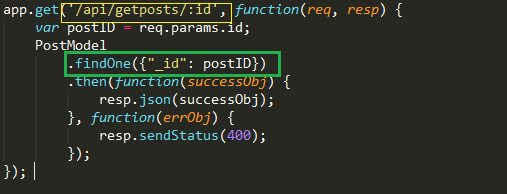
* Service will have following code
* editPost() will inturn call to this.getAllPosts(postID) to receive the requested data but this time it would pass postID to getAllPosts()
* getAllPosts() will then append this postID to the url that is to be mapped in server.js



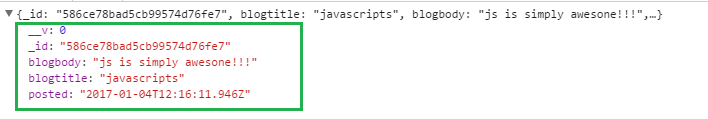
* getAllPosts () will have following modification



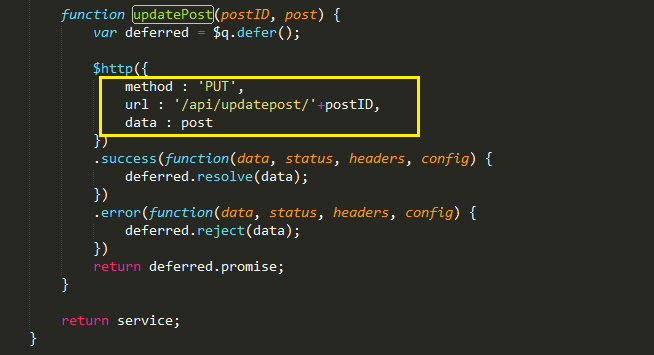
* url in this case would be **api/getposts/{postID}** which is to be mapped in server js file as **app.get**(**'/api/getposts/:id**', function(){…..} )
* as we know **id** will be unique for every post in mongodb instead of find() method of mongo we call findOne(), this will return us exact object containing blog data of requested postID
* also .find() would return us array while .findOne() would return us exact object



* if success above code will return us object as



* this object then is sent back to service and then to controller to fill the textboxes saying this object is to be edited
* now in order to edit/update the edited data we create another button named Update which would fire **updatePost(post.\_id, post)** this would send unique ID and edited data to database
* controller calls service and service would send this updated data to server.js file by calling **PUT** method of $http object



* the above url is mapped to **/api/updatepost/{postID}** from server.js file
* var postTitle = **req.body.blogtitle**; : gets data from request payload

var postBody = **req.body.blogbody**; : gets data from request payload

* sends this updated data to database by calling **update** method of mongo
* .update({**id**}, {**data**}) will accept 2 params – first is unique id and second is updated/edited data
* If success it will send updated data back to service and then to controller which in tur call getAllPosts() and this will then update the UI

