**Build IO**

EJS, Strongloop, HTML, CSS, Authentication & authorization

Data source integration, git

**What is EJS templets etc**

### **EJS -** [**Embedded JavaScript templates**](https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&sqi=2&ved=0ahUKEwih6JKu1KHVAhVJ1GMKHSMUDt8QFggpMAI&url=http%3A%2F%2Fejs.co%2F&usg=AFQjCNGJj1QqCdRpVlPfOqGdZDUrOe42BA)

 is a simple templating language that lets you generate HTML markup with plain JavaScript.

(This is to display the output came from database on the browser with some proper format .e.g create table OR list )

A **template engine** enables you to use static template files in your application. At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client. This approach makes it easier to design an HTML page.

Some popular template engines that work with Express are [Pug](https://pugjs.org/api/getting-started.html), [Mustache](https://www.npmjs.com/package/mustache), and [EJS](https://www.npmjs.com/package/ejs). The [Express application generator](https://expressjs.com/en/starter/generator.html) uses [Jade](https://www.npmjs.com/package/jade) as its default, but it also supports several others.

 Jade has been renamed to [Pug](https://www.npmjs.com/package/pug).

To render template files, set the following [application setting properties](https://expressjs.com/en/4x/api.html#app.set), set in app.js in the default app created by the generator:

* views, the directory where the template files are located. Eg: app.set('views', './views'). This defaults to the views directory in the application root directory.
* view engine, the template engine to use. For example, to use the Pug template engine: app.set('view engine', 'pug').

Then install the corresponding template engine npm package; for example to install Pug:

npm install ejs --save

app.set('view engine', 'ejs')

Now Let’s first create an index.ejs file within the views folder so we can start populating data.

<ul class="quotes">

<% for(var i=0; i<quotes.length; i++) {%>

<li class="quote">

<span><%= quotes[i].name %></span>

<span><%= quotes[i].quote %></span>

</li>

<% } %>

</ul>

In EJS, you can write JavaScript within <% and %> tags. You can also output JavaScript as strings if you use the <%= and %> tags.

Once the view engine is set, we can begin generating the HTML with our quotes. This process is also called **rendering**. We can use the render object built into the response object render to do so. It has the following syntax:

res.render(views, locals)

**The first parameter, views**, is the name of the file we’re rendering. This file must be placed within a views folder.

**The second parameter, locals**, is an object that passes data into the view.

app.get('/', (req, res) => {

db.collection('quotes').find().toArray((err, result) => {

if (err) return console.log(err)

// renders index.ejs

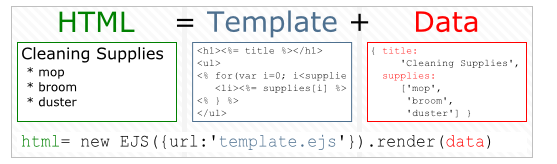
res.render('index.ejs', {quotes: result})

})

})

**67 What is rendering in node js**

res.render() function compiles your template inserts locals(data) there, and creates html output out of those two things.



The render method works when you have a templating engine in use such as handlebars.js or jade OR EJS.

A templating engine is a node module associated with express (which some people refer to as an express plugin) which parses the template file and generate the HTML output.

The **sendfile** method simply sends the file to the client.

Since you are using an HTML file, there is nothing particularly to be parsed by the templating engine. So, the output of render is same as that of sendfile (i.e., the HTML written in the file). Hence, both produce the same result.

app.get('/',function(req,res){  
  res.**sendFile**('index.html');  
  *//It will find and locate index.html from View or Scripts*  
});

app.get('/', function(req, res) {

res.**render**('index.html');

});

**HTML DOM -**

**Document Object Model** is a wayto access and modify the document.

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page

The Objects are organized in a hierarchy. Top window object , then Doument object and then its childs.

Strongloop

55 What is strongloop?

Strongloop is the (IBM) company that has built an API Platform which features the open source Loopback framework. Loopback enables you to quickly compose APIs and runs on top of the express framework. It could've been named Strongloop Loopback Starter.

See: <https://strongloop.com/node-js/loopback-framework/>

In addition to the loopback framework, the Strongloop API Platform also includes the Arc graphical UI, which has tools for building, profiling, and monitoring Node Apps.

You could create your API using Loopback & then monitor & profile that API using Arc. Both are a part of the strongloop platform.

# **56. What is** [**slc arc**](https://docs.strongloop.com/display/NODE/slc+arc)

It’s a GUI for all strongloop products

your application will launched as a node cluster(having parent child process ) for production

**StrongLoop** Arc is a graphical tool for building, deploying, managing, and monitoring **LoopBack** applications and APIs. It enables you to: Create and modify **LoopBack** models and datasources.

Run [StrongLoop Arc](https://docs.strongloop.com/display/APIS/Using+Arc), by default opening it in a web browser window

Slc loopback

Give app name

Cd app directory

Here run slc arc

Go ro browser and use this GUI to create modules and other stuff.

StrongLoop Arc will use a different port number each time you run it. To run it on a specific port, use the PORT environment *variable, for example:*

|  |
| --- |
| $ PORT=4000 slc arc |

[*https://www.youtube.com/watch?v=ckD8KlZMl0k*](https://www.youtube.com/watch?v=ckD8KlZMl0k)

57. Folder structure of strongloop app.

Client

Common --models

Node\_modules

Server - boot

Package.json

Readme.md

-🡪 when slc arc is used one more folder is created [ .strong-pm]

**strong-pm** is a process manager that manages cluster size.

slc:is application generator

'loopback-connector-mongodb ' 🡪 Connector for strongloop

'loopback-connector-mysql ' 🡪 Connector for strongloop

7. What is loopback

**LoopBack** is an open source Node.**js** framework built on top of Express

It is faster to do project using Loopback than express

It generates API code automatically(folder structure), makes design and testing easier and so that developers can focus more on user experience and business logic. It simplifies connecting to Oracle/MongoDB/SQL Server/MySQL legacy and new data.

[difference between ***authentication*** and ***authorization***](https://www.google.co.in/search?rlz=1C1GGRV_enIN761IN761&q=difference+between+authentication+and+authorization&spell=1&sa=X&ved=0ahUKEwjps4mF68zXAhXDuI8KHQlICzwQvwUIIigA)

When you log on to a PC with a user name and password you are authenticating.**Authorization** is the process of verifying that you have access to something.

**React**

ReactJS is JavaScript front-end library used for building reusable UI , developed by Facebook components.

It is used for handling the view layer for web and mobile apps. ReactJS allows us to create reusable UI components.

## React Features

* **JSX** − JSX is JavaScript syntax extension. It isn't necessary to use JSX in React development, but it is recommended.

**OAuth 2.0**

**Bearer Token – Is OAuth 2.0 subpart**

Used to access protected resources

OAuth 2.0 - Accessing a Protected Resource

They are included in the authorization header as follows −

Authorization: Bearer [token-value]

OAuth 2.0 is a very flexible protocol that relies on SSL (Secure Sockets Layer that ensures data between the web server and browsers remain private) to save user access token.

OAuth 2.0 is a simple protocol that allows to access resources of the user without sharing passwords.

It allows sharing of resources stored on one site to another site without using their credentials.

Following are the parameters and their descriptions.

* **client\_id** − It should be set to the client id of your application.
* **redirect\_uri** − It should be set to the URL. After the request is authorized, the user will be redirected back.
* **response\_type** − It can either be a code or a token. The code must be used for server side applications, whereas the token must be used for client side applications. In server side applications, you can make sure that the secrets are saved safely.
* **We use scope also**

https://publicapi.example.com/oauth2/authorize?client\_id=your\_client\_id&redirect\_uri=your\_url

&response\_type=code

**Mongo Coding**

var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost:27017/Employee';

var Connection = function(){};

exports.Connection = Connection;

Connection.prototype.getConnection = function(url,callback){

console.log('Insile getConnection');

MongoClient.connect(url,function(err,db){

if(err){

console.log('unable to connect to db'+err);

callback(err,null);

}else{

callback(null,db);

}

});

}

// Method implementation

var **getPassword** = function(req,callback){

console.log('Inside getPassword');

var name = req.body.name;

var num = req.body.DOB;

var subName = name.substring(0, 3);

var subNum = num.substring(0, 2);

var password = subName+subNum;

console.log('password at first place'+password);

callback(null,password);

}

var **getConnection** = function(req,callback){

console.log('Inside getConnection');

MongoClient.connect(url, function(err, db) {

if(err){

console.log('unable to connect to db'+err);

callback("fail",null);

}

else{

console.log("MISSION ACCOMPLISHED"+db);

callback(null,db);

}

});

}

var **createCollection** = function(db,callback){

console.log('Inside createCollection');

db.createCollection('newEmployee', function(err, collection) {

if(err){

console.log('Error creating collection ...');

callback('fail',null);

}

else{

console.log('collection creartion successful...'+collection);

callback(null,collection);

}

});

}

var **postData** = function(db,collectionData,reqData,callback){

console.log(" Inside postData Method");

db.collection('newEmployee').insert(reqData,function(err, obj) {

if(err){

console.log('Error creating collection ...');

callback('fail',null);

}else{

console.log(" document inserted");

callback(null,obj);

}

});

};

var **getData** = function(db,callback){

console.log(" Inside getData Method");

db.collection('newEmployee').find({}).toArray(function(err,result){

if(err){

console.log('Error in getting data '+err);

callback('fail',null);

}else{

console.log('data fetched successfully.....'+result);

console.log('data fetched successfully.....'+result.length);

callback(null,result);

}

});

};

var **postArray** = function(db,dataArray,callback){

db.collection('newEmployee').insertMany(dataArray,function(err, obj) {

if (err) throw err;

console.log(" document inserted");

callback(null,obj);

});

};

var **deleteData** = function(db,callback){

db.collection('newEmployee').remove(function(err, obj) {

if (err) throw err;

console.log(obj.result.n + " document(s) deleted");

callback(null,obj);

});

};

app.listen('3000',function(){

console.log('listening...........');

});