## What is node Js

* Open Source, a cross-platform runtime environment for executing code outside the browser
* Used to build backend services such as API
* Node is used to build Highly scalable, data-intensive and real-time apps
* It not framework or programming language

## Why do we use node Js instead of using other backend languages?

* Easy to use
* Great for prototyping and agile development
* Superfast and highly scalable services
* Use of javascript
* Cleaner and more consistent codebase
* Large ecosystem of open-source library
* Paypal found that using node you can built services
  + twice fast
  + 33% less code
  + 40% fewer files
  + 2x request/sec
  + 35% faster response as compared to java

## Architecture of node

* Before node, we use JS to build applications inside the browser different browsers have different javascript engines like
  + Internet explorer Chakra
  + Mozilla Firefox spider monkey
  + Chrome V8
* These engines convert javascript to machine code so due to different engine code can behave differently in the browsers
* Ryan dahl came up with the idea to run javascript at run time environment so he used google V8 engine inside cpp code called it node.exe
* It has also some objects that provide an environment for javascript code different from browsers
* It gives us additional modules not available in the browser we can use node to access file system and network as well
* We have objects like document and window in the browser but don’t have in the node

## How node works

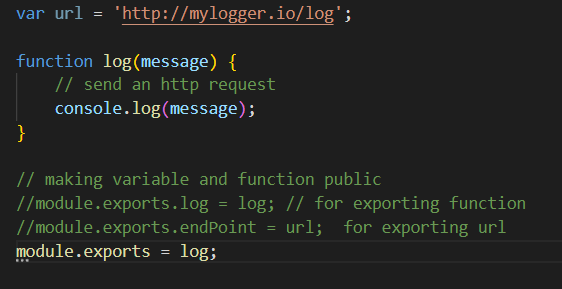
* Node has non-blocking asynchronous architecture
* Means a single thread can handle multiple requests
* If node was using synchronous architecture mean one thread will serve a single request at a time which means it will consume resources

## Node Module System

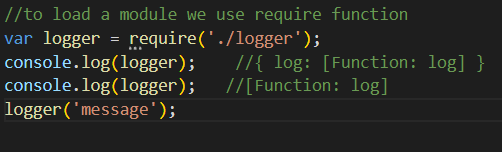
### Global objects

* In the browser the objects mentioned below are defined inside windows object i.e. window.console.log() similarly in the node these objects are defined inside a global object i.e global.console.log()
  + console.log()
  + setTimeout()
  + clearTimeout()
  + setInterval()
  + clearInterval()
* If we define a variable or function it is defined inside global scope but we can access it using global object it will give undefined
* We should not use global scope with variable or function because suppose you have defined two function of same name in two different files when first function is loaded in global scope and second file is called the variable is overwritten can cause issue so avoid instead use modules
* Every file in the node is considered a module. Variables and functions defined in the file are considered private or in the scope of that module if you want to use a variable outside of the module you have to export it

### Creating module

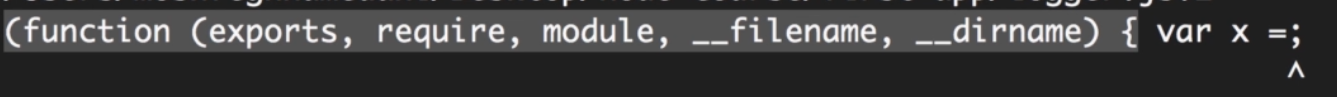


### Loading module



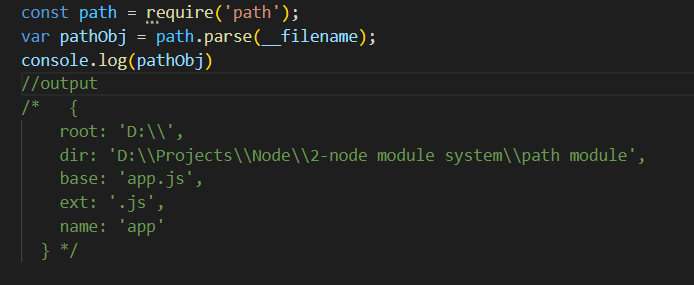
### Module Wrapper Function

* Node doesn’t execute program directly its wrap inside a function
* Immediately invoked function expression
* Require argument appears to be global but it not its local to each module
* Export is shortcut or reference to module.export can’t be changed
* Filename and dir name represent filename and directory



There are some useful built in node modules

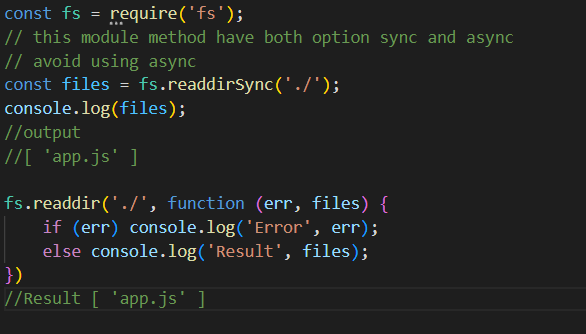
### Path module



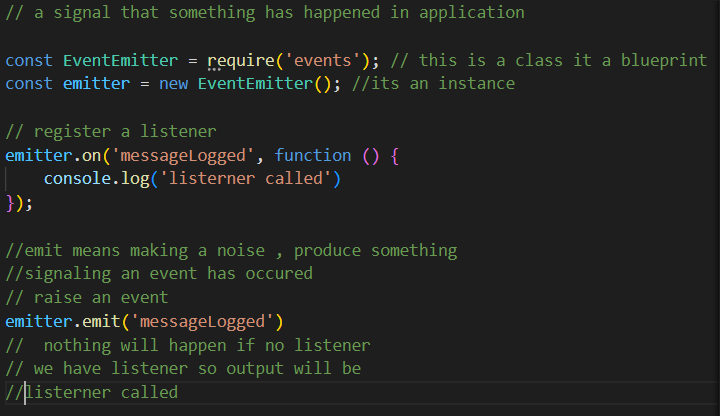
### Os module



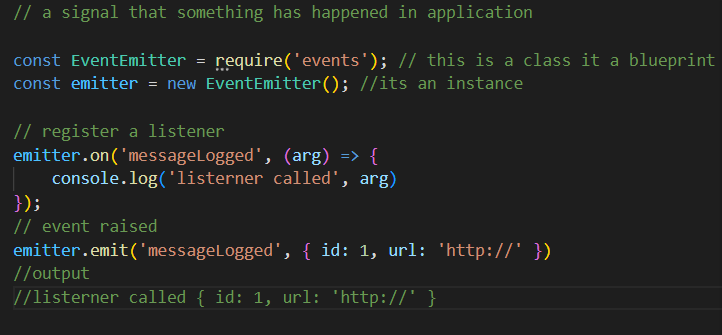
### File system module



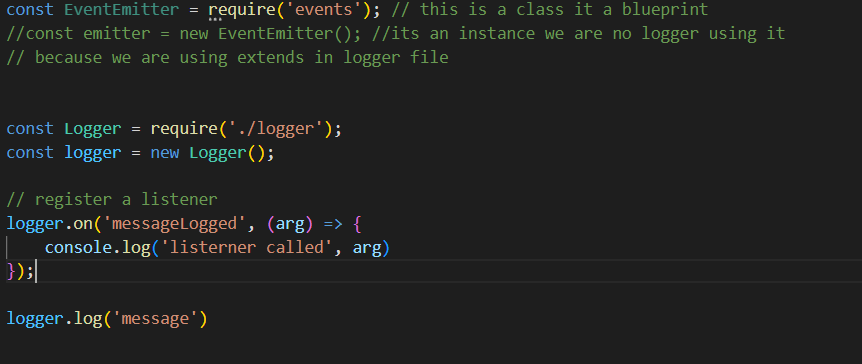
### Event Module

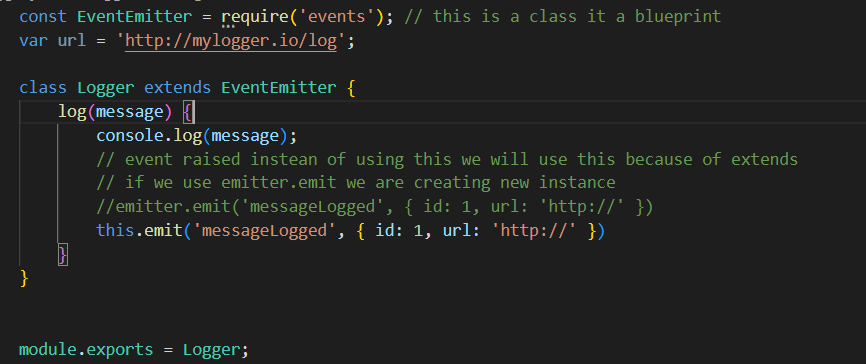


### Event arguments



### Extend event module





### HTTP module

## 

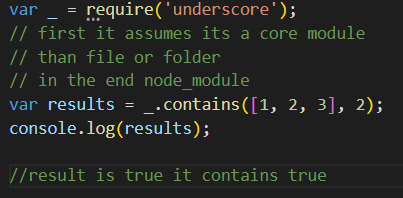
## Node Package Manager

* Command line tool
* Registry of 3rd party library that we want to our application

### Package.json

* Json file that include basic information
  + Name
  + Version
  + Authors
  + Location of git repository
* It’s basically is metadata about your application
* To create json file in the folder run **npm init** command
* Every node package has pakage,json file

### Using a package



### Installing mongoose

* Mongoose is a [**MongoDB**](https://www.mongodb.org/) object modeling tool designed to work in an asynchronous environment. Mongoose supports [**Node.js**](https://nodejs.org/en/) and **[Deno](https://deno.land/)** (alpha)
* npm i mongoose

### Semantic Versioning

* “mongoose’: “^4.13.6”
* Also called SemVer
* It has 3 components **4.13.6 (Major Version. Minor Version . Patch Release)**
* Caret ^ indicates interest in newer minor version ^**4.13.6**  greater than 4.x
* Tilde ~ indicates interest in newer patches version ~**4.13.6**  greater than 4.13.x

### Versioning of node modules

* npm list
* npm list –depth=0 // to only list dependencies of your applications

### View dependencies of a module

* **npm view mongoose** this will display package.json file of the module
* **npm view mongoose dependencies** this will only display dependencies of the module
* **npm view mongoose versions** this will only display all versions released of mongoose module

### Installing specific version of node module

* npm install [mongoose@2.4.2](mailto:mongoose@2.4.2)

### Updating local packages

To find which packages are outdated

* npm outdated
* npm –g outdated

To update app

* **npm update** this will not change the major version will only change the minor version and patch
* **npm install –g npm-check-updates** to install npm check updates module
* **npm-check-updates** to list only outdated modules
* ncu –u (short form of **npm-check-updates upgrade**) this command will also change the major version of module as well.

### Development dependencies

Development dependencies are used on DEV environment not to install on PRD environment

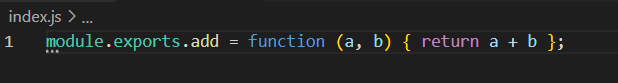
* npm install jshint --save-dev

### Uninstalling a package

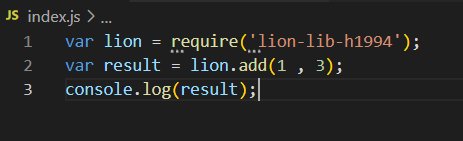
* To uninstall **npm uninstall mongoose** or **npm un mongoose**
* To uninstall global modules **npm un –g**

### Publish a package

* Create a folder
* Goto folder write npm init –yes
* Add index.js file which in the entry point of our module



* If you don’t have an account on npmjs.org npm adduser
* If you have an account npm login
* When logged in write npm publish
* Now goto project where you want to add your module in my case **npm install [lion-lib-h1994](https://www.npmjs.com/package/lion-lib-h1994" \t "_self)**
* Using the package



### Updating the package

* Goto module cmd write npm version you will see current version
* To update major version **npm version major**
* To update minor version **npm version minor**
* To update patch version **npm version patch**
* **Npm publish**

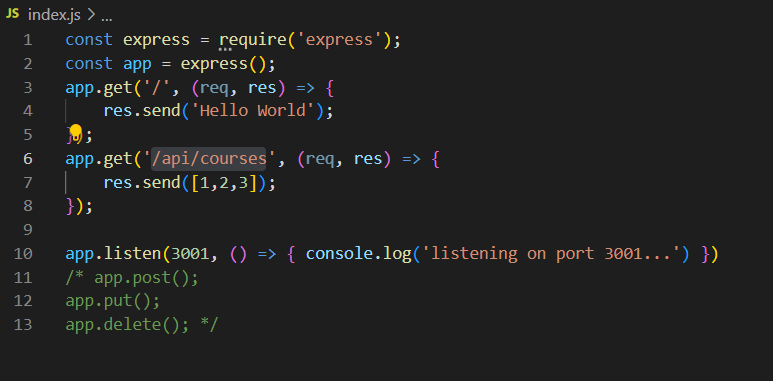
## Express JS

### Restful API’s

* Client-Server Architecture
* App is the client or front-end
* Client needs the backend to get or save data
* Communication between client and server happens using the HTTP protocol
* On the server we expose a bunch of services that are accessible via the HTTP protocol client access the services using an HTTP request
* Rest (representational state transfer) convention to build the HTTP services
* We used simple HTTP operations for CRUD (create, read, update, delete)
* At server we expose service (Endpoint) for client to talk to server.
* Endpoint consist of:
  + **http://vidly.com/api/customers**
  + **http://** for not secure channel a
  + **https://** for secure channel
  + **vidly.com** domain of application
  + **/api/** this is not necessary but many companies follows this convention to expose Restful services
  + **/customers** which is a resource

### Intro of Express Js

* We don’t use the HTTP module in complex applications because of the large number of APIs we can’t hardcode API’s
* Express is a fast and lightweight framework
* Express provides proper structure we can easily create more routes while keeping our code maintainable.
* Authentication is out of scope to express js because lightweight framework



### nodemon

* nodemon stands for node monitor instead of reloading this automatically reload the app
* install using **npm i -g nodemon**
* run app using **nodemon index.js**

### environment variable

* An environment variable is **a dynamic-named value that can affect the way running processes will behave on a computer**



* On Cmd write command to set command of your node app
  + On windows **set PORT=5000**
  + On mac **export PORT=5000**

### Route parameter

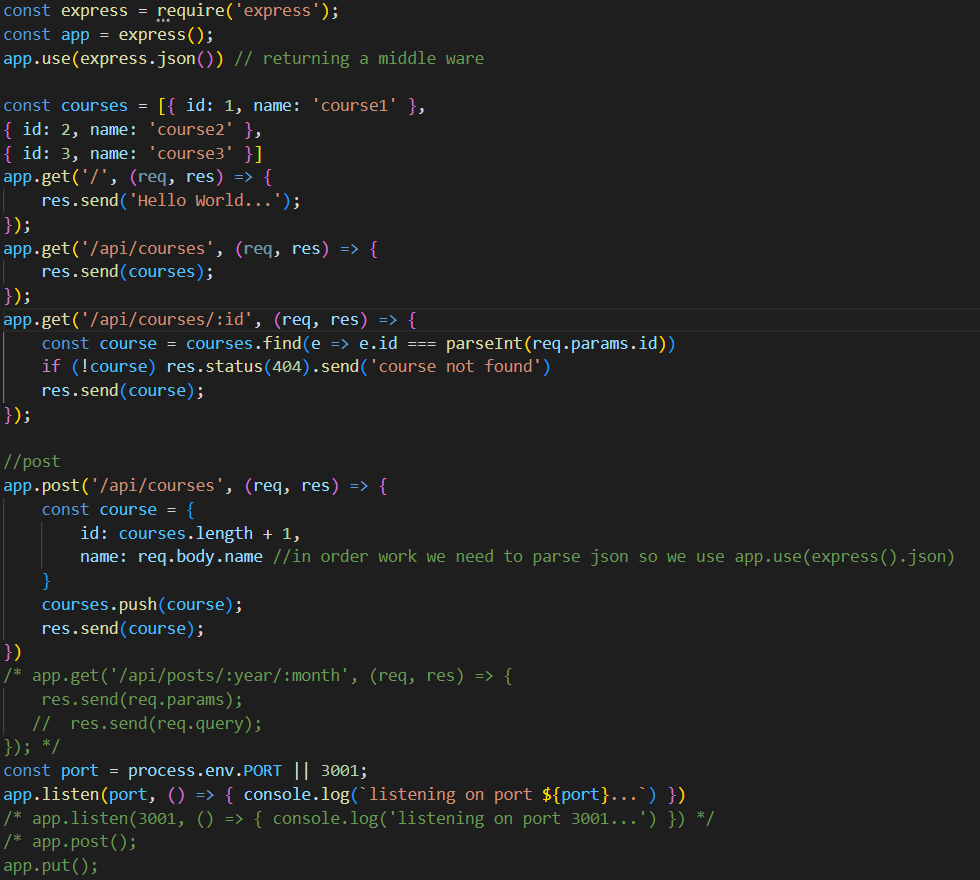
* /api/courses/:id



### Handling HTTP Get Requests

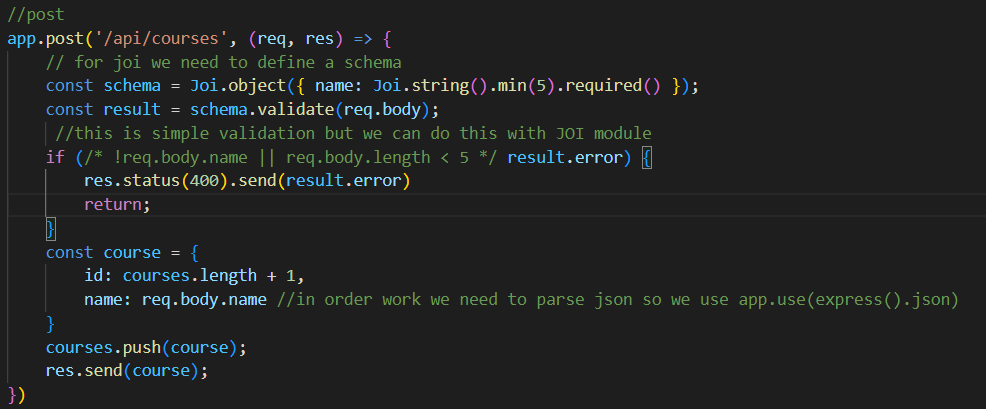


### Handling HTTP Post Requests

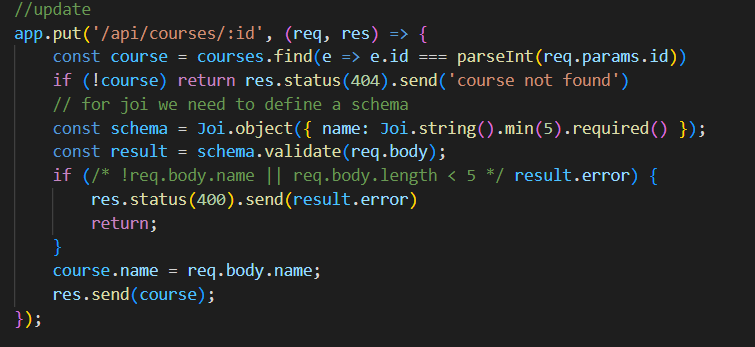


### Input Validation

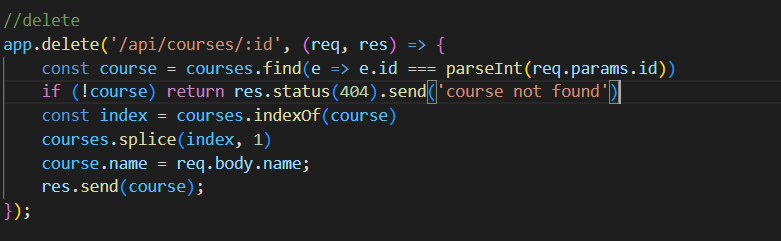
You should never trust the user you must always do input validation



### Handling HTTP Update Requests



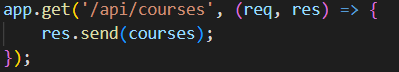
### Handling HTTP Delete Requests



## Advanced Express JS

### Middleware

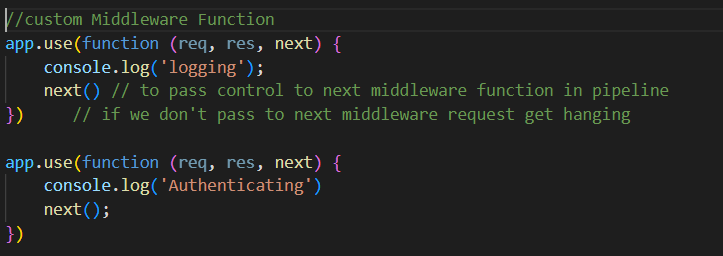
Is a function that takes a request object passes a response to a client or passes it to another middleware function. The example mentioned below is a middleware function because it takes a request and returns a response



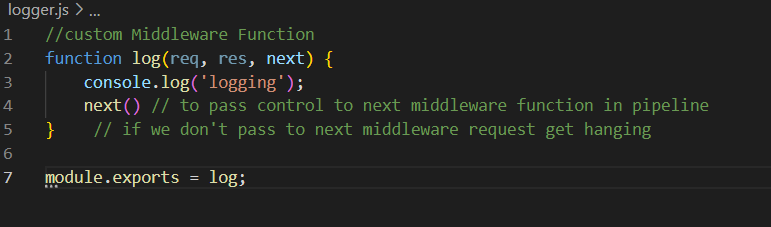
Another example is mentioned below return a middleware function the job of this function is to read the request if there is JSON object in the body of the request it will parse it into a JSON object and sat it into req.body

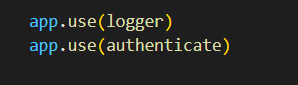


### Custom Middleware Function



Normally we don’t define custom middle ware like this we create a separate file and use it like this





### Built-in middleware

* urlencoded(): to parse the body of requests with URL-encoded payload (deprecated)



* static(): to serve static files



This is used to access static content in the example above you can access the content of public folder.

### Third-party middleware

* helmet() help secure express apps with various HTTP headers

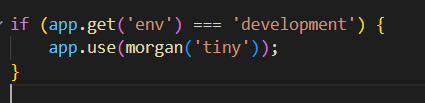


* morgan() to log HTTP request



### Enviroment

* To console the current environment
  + **process.env.NODE\_ENV**
  + **app.use(‘env’)**

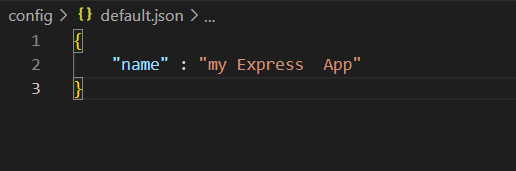
****

* to set the environment write on console
  + **export NODE\_ENV=production**

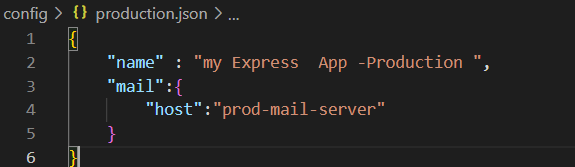
### Configuration

Install config or rc package

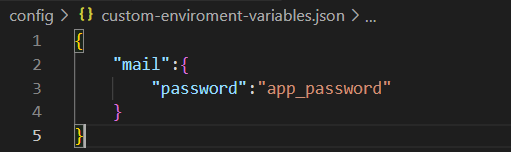
* npm i config

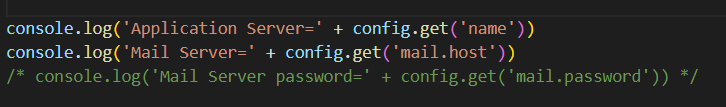






* To set app\_password go to console and write **set app\_password**



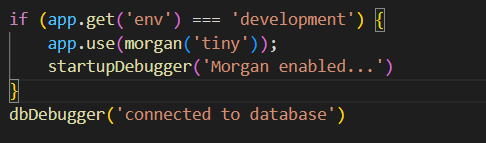


### Debugging

* Install module npm i Debug
* Alternative of console.log()
* Declare like this

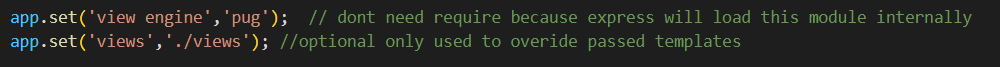


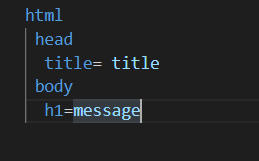
* Use is like this



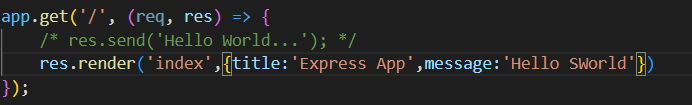
* No output will be displayed until you set debug on console like
  + **set DEBUG=app:startup**
  + **set DEBUG=app:startup,app:db**
  + **set DEBUG=\* this is a wildcard will include all namespaces**
  + **DEBUG=app:startup nodemon index.js**

### Templating Engines

* Sometimes you have to return HTML markups to the client there we use templating engines
  + Pug
  + Mustache
  + EJS
* 
* Create views folder now and create index.pug file

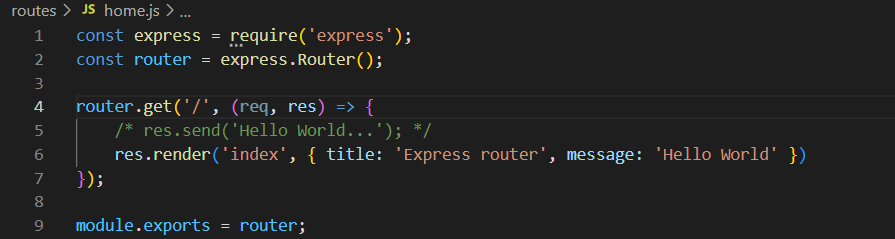


* Now where you want to render this template

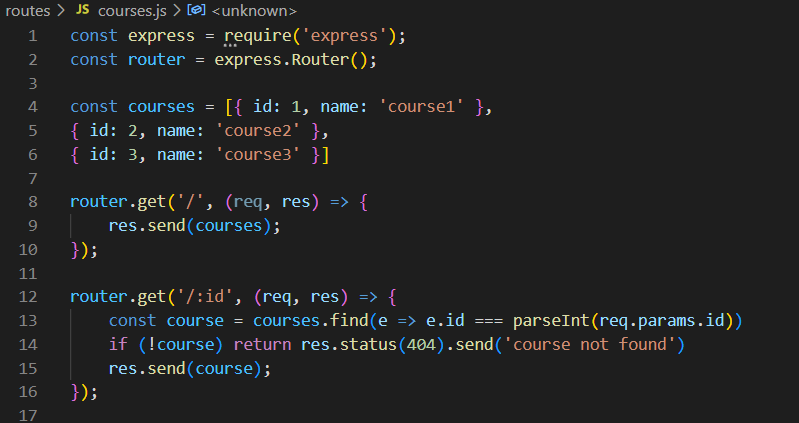


### Restructuring the code

* we should route each api in separate file
* file for home “/” route



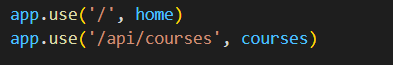
* file for courses route same as home small example in end of file **module.exports=router;**



* in index file declare the files

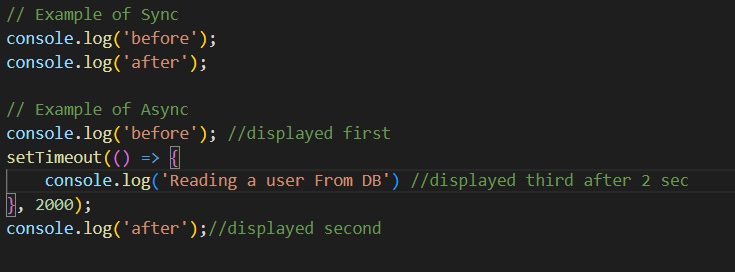


* Call the Api’s (in the courses file we are not using the full path because already declare here in index.js file)



## Asynchronous JavaScript

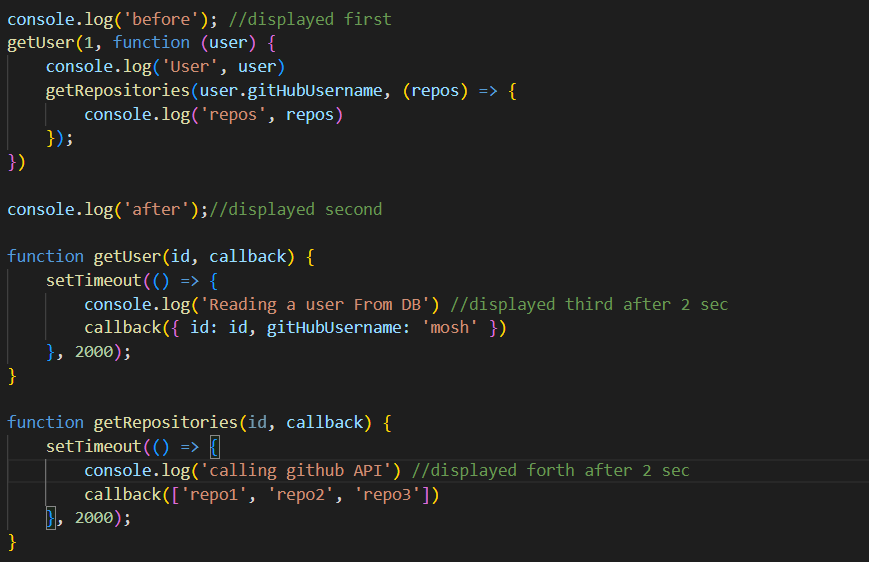
### Example of Sync and Async



### Pattern for dealing with Async Code

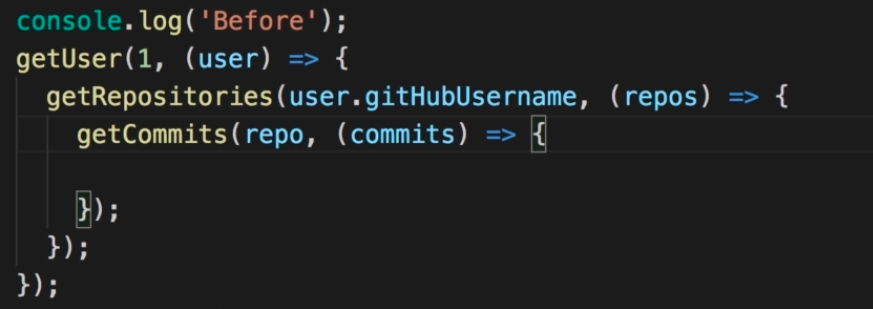
* Callbacks
* Promises
* Async/await

### Callbacks



### Callback hell or Christmas tree

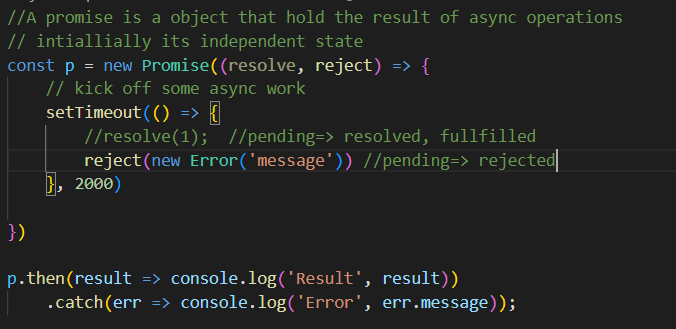
* Code can contain many nested callbacks
* Which can difficulty in understanding the code



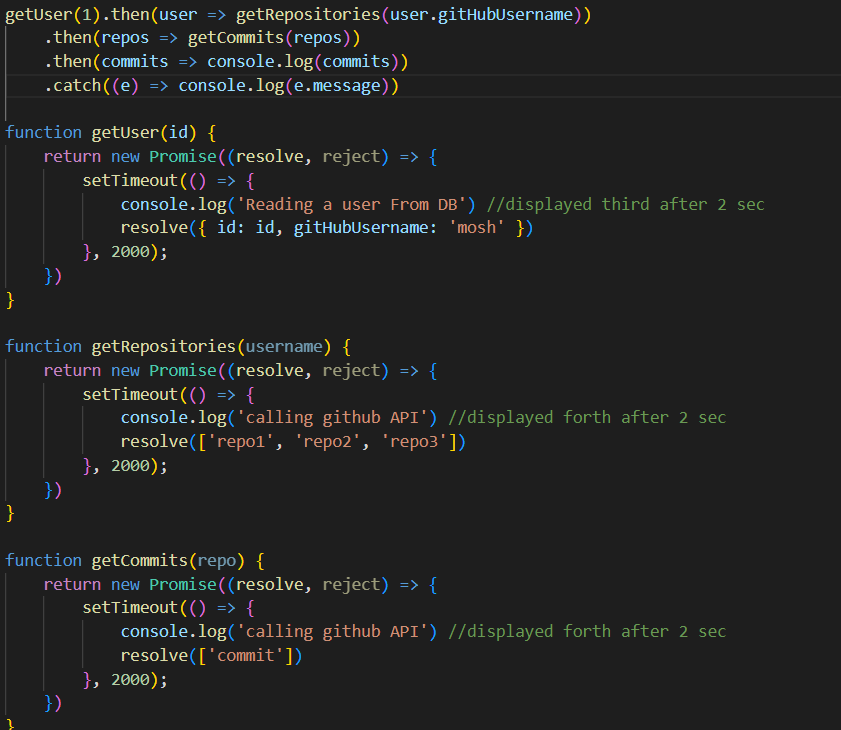
### Small solution to callback hell named Function



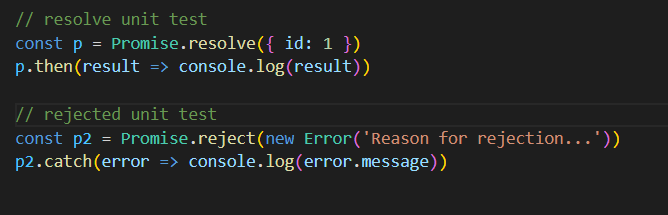
### Promise



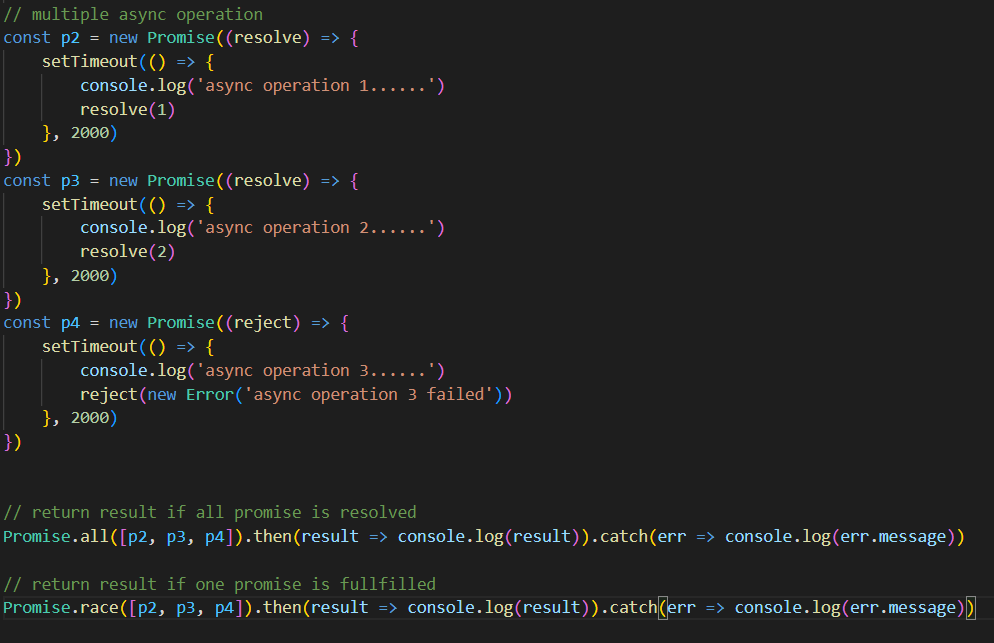
### Consuming Promise



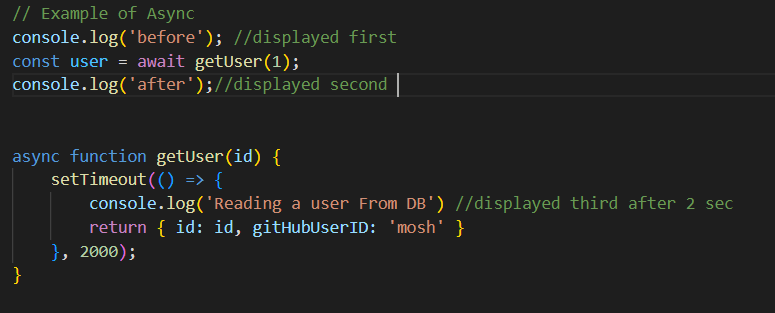
### Settled promise



Running parallel promise



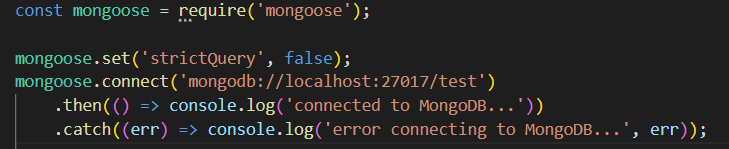
### Async and await



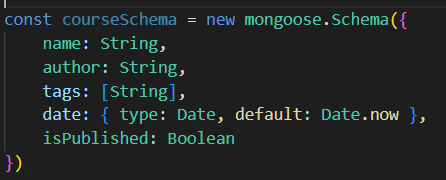
## Mongo DB

* Document DB or NoSQL DB
* Don’t have the concept of tables, schema, record and columns
* Simply store json object in DB

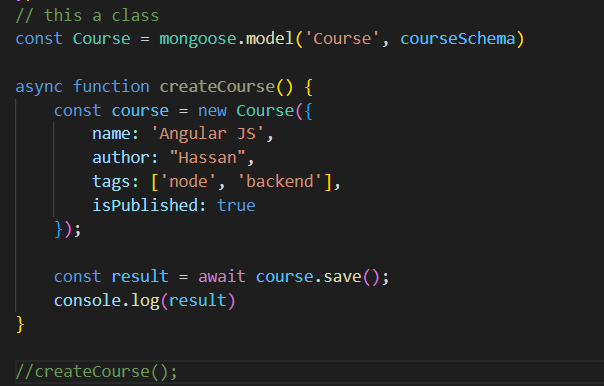
### Creating Connection



### Creating Schema



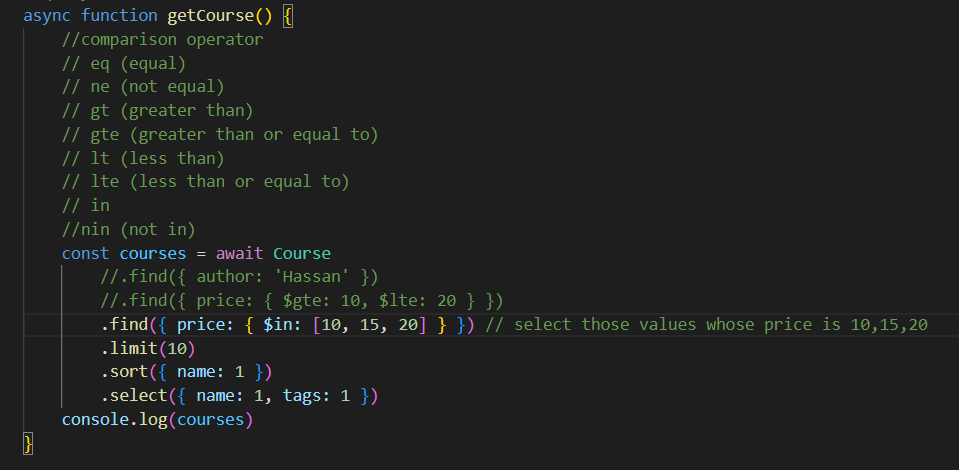
### Models



### Querying Documents



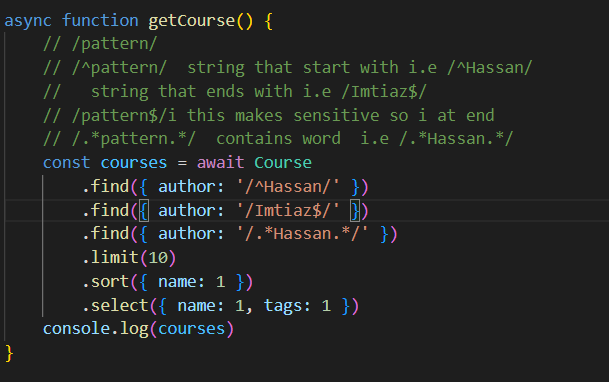
### Comparison Query Operator



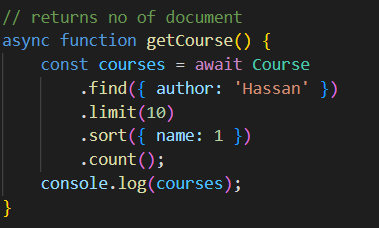
### Logical Expressions



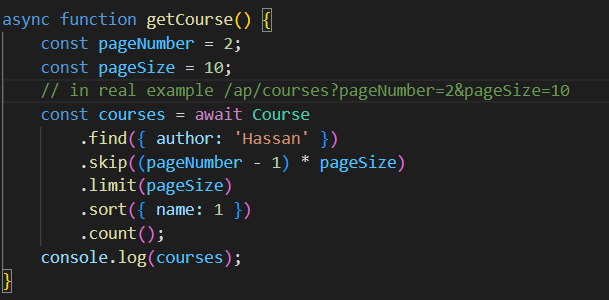
### Regular Expressions



### Counting

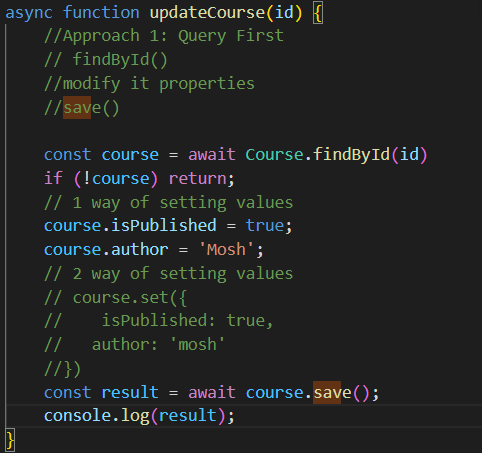


### Pagination

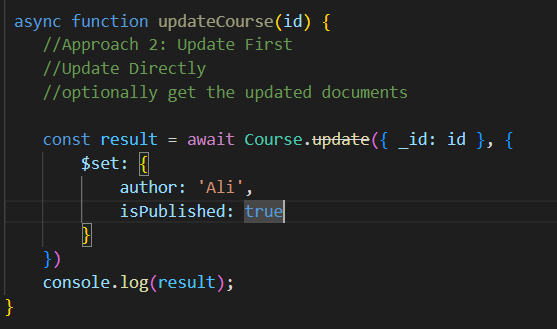


### Updating Document

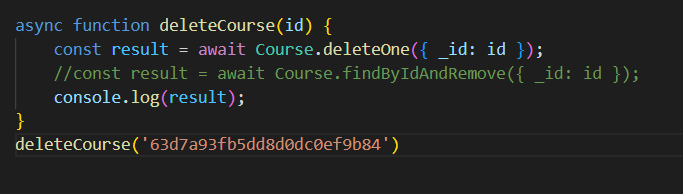
#### Approach 1



#### Approach 2



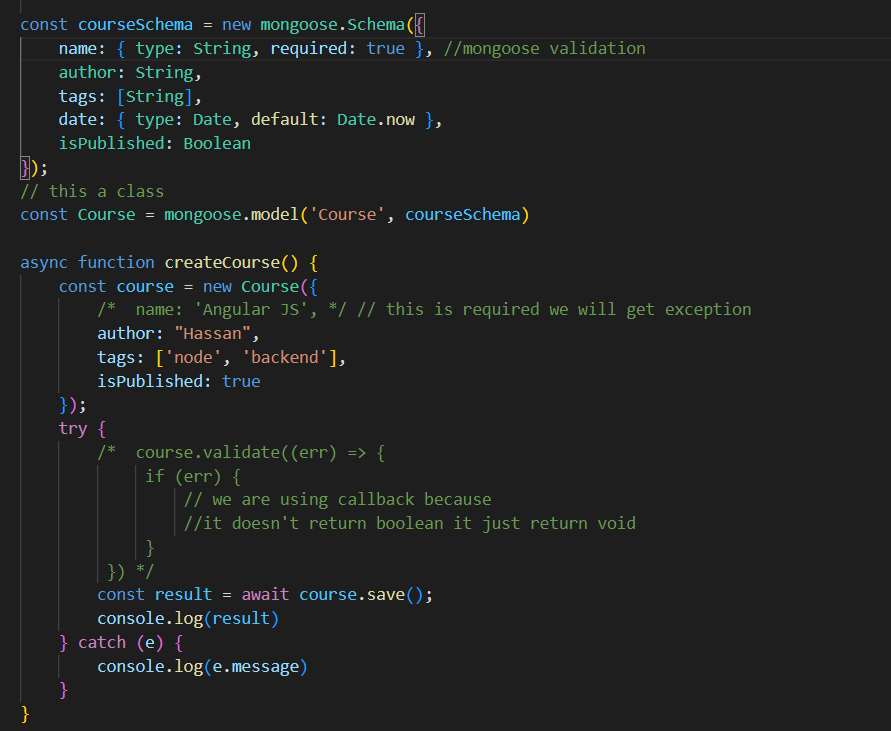
### Delete



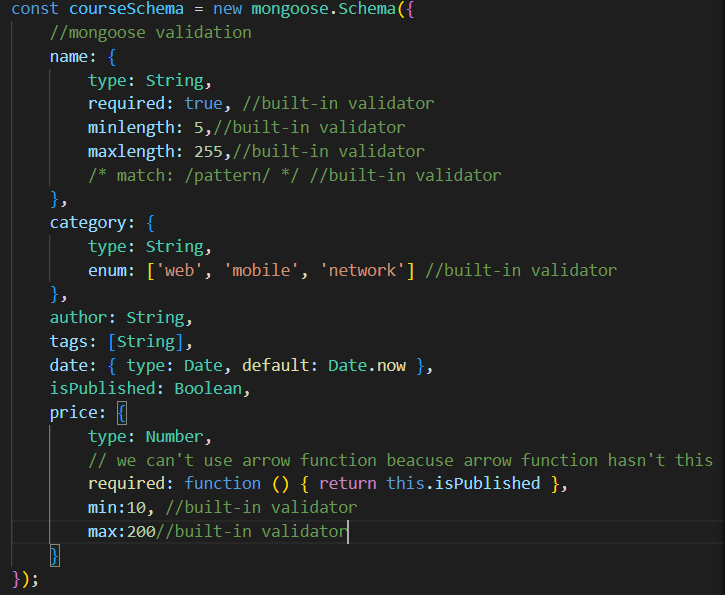
## Mongoose Data Validation

* We should use a combination of mongoose and joy node module for data validation
* We use joy in restful apis
* We use it as first attack to make sure the user is sending valid data and we use mongoose validation to ensure we are sending data to mongoose in a right shape

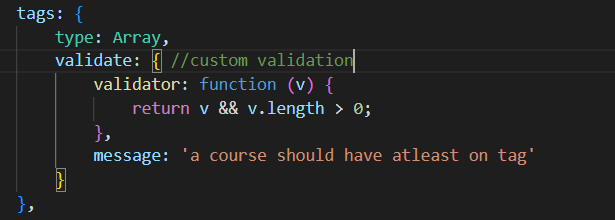
### Validation



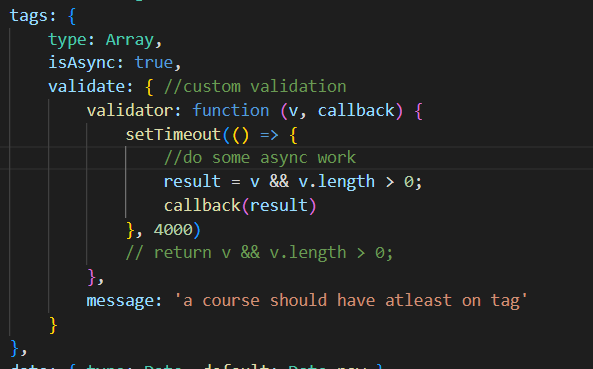
### Built-in validators



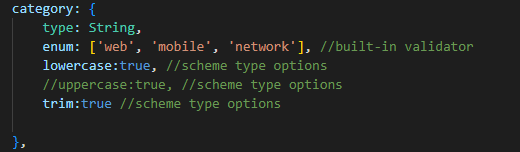
### Custom validators

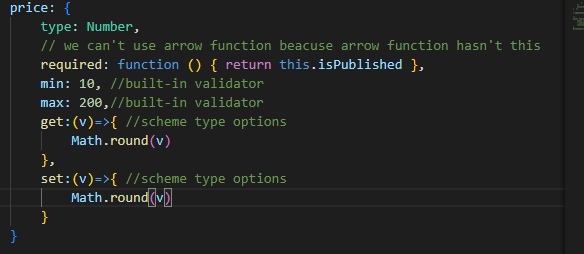


### Aysnc validator



### Scheme type options

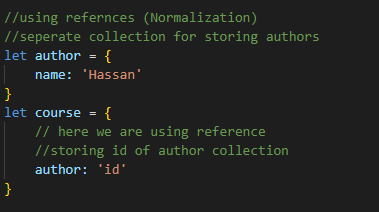




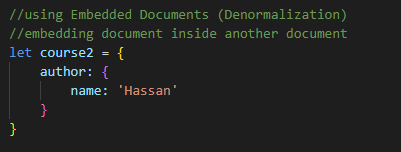
## Mongoose- Modeling Relationships Between Connected Data

### Modeling and relationship

* In RDMS we have e concept of the relationship which enforces data integrity
* In MongoDB or NoSQL we don’t have a relationship
* There are three approaches to solving a relationship issue
  + Normalization – using references (creating separate documents and referencing)



* + Denormalization – using embedded document (embedding document inside a document)



* + Hybrid approach suppose each author has 50 properties and we don’t want to duplicate those properties in the courses document so we have a separate collection of authors instead of referencing in author we can embed the author document inside the course document **but not whole document**



* There is an issue with these approaches we have a tradeoff between consistency or performance
* In normalization there is consistency if we want to make change in the author name we make one place it will automatically reflect in other places where the trade-off is the performance in this approach we have to do additional query
* In denormalization we don’t have to make additional queries suppose we make a change in an author name data in other places will not update and will create inconsistencies

### Transactions

* In RDMS we have the concept of transactions which mean group of queries performing operations which mean if one query failed whole transaction failed which mean data will roll back to initial state
* Fawn library allow you to do Two phase commit