PAGE :
DATE: / /

MEAN Final Prep. →

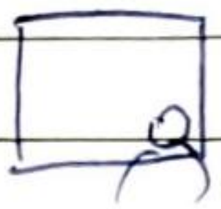
Unit I Web Development with MEAN Full stack →

Q- ~~Benefits of Full stack?~~

⇒

Q- Intro to MEAN?

⇒

Database	Application server	front end
		
MongoDB	NodeJS & Express Lang: JavaScript	Angular Lang: TypeScript

* Node JS →

- NodeJS is N in MEAN. It is a software platform that allows you to create your own web server & build web applications on top of it.
- NodeJS isn't a itself web server, neither is it a language. It contains a built in HTTP server library
- NodeJS can be written in JavaScript
- NodeJS is extremely fast & makes efficient use of system resources.

PAGE :
DATE: / /

- Node JS server is a single threaded, rather than giving each visitor a unique thread & a open separate resources, the server has every visitor join the same thread.

Express - The framework →

- Express is the E in MEAN. Express is a web application framework for Node JS that designed to perform the tasks in a well-tested repeatable way.

- Express sets up a web server to listen incoming requests & return relevant responses.

- One of the greatest features of Express is that it provides a simple interface for directing an incoming URL to a certain piece of code.

- Express provides support for many templating engines that make it easier to build HTML pages in an intelligent way.

Mongo DB - The database →

- NO SQL database
- MongoDB is M in the MEAN which stores the document in non relational way.

- MongoDB stores documents as BSON (Binary JavaScript Object Notation)

- Ex
{
"first Name" : "Phil",
"last Name" : "Dunphy",
_id : "0521afe32d072"
}

- mongoose with mongoDB helps in data modeling. Data modeling is defined as what data must be in a document.

- mongoose also makes it easier to manage the connections to mongoDB database & to save & read data.

Angular →

- Angular is A in MEAN. Angular is a Javascript framework for creating the interface for your website or application.

- Angular is specially designed for single page application. In SPA, everything runs inside the browser & never does a full page reload.

- Angular applications are written in typescript which is a superset of Javascript.

- Angular also supports for two-way data binding.

PAGE :
DATE: / /

Supporting cast ->

- **Git for source control ->**

- Git is a distributed revision control & source code management system that allows several people to work on the same codebase at the same time on different computers & networks

- These can be pulled together, with all changes stored & recorded. It's also possible to roll back to an earlier state if necessary.

- **Bootstrap ->**

- Bootstrap is a front end framework that provides a wealth of help for creating a great user interface

- Bootstrap provides a responsive grid system, default styles for many interface components, & the ability to change the visual appearance with themes.

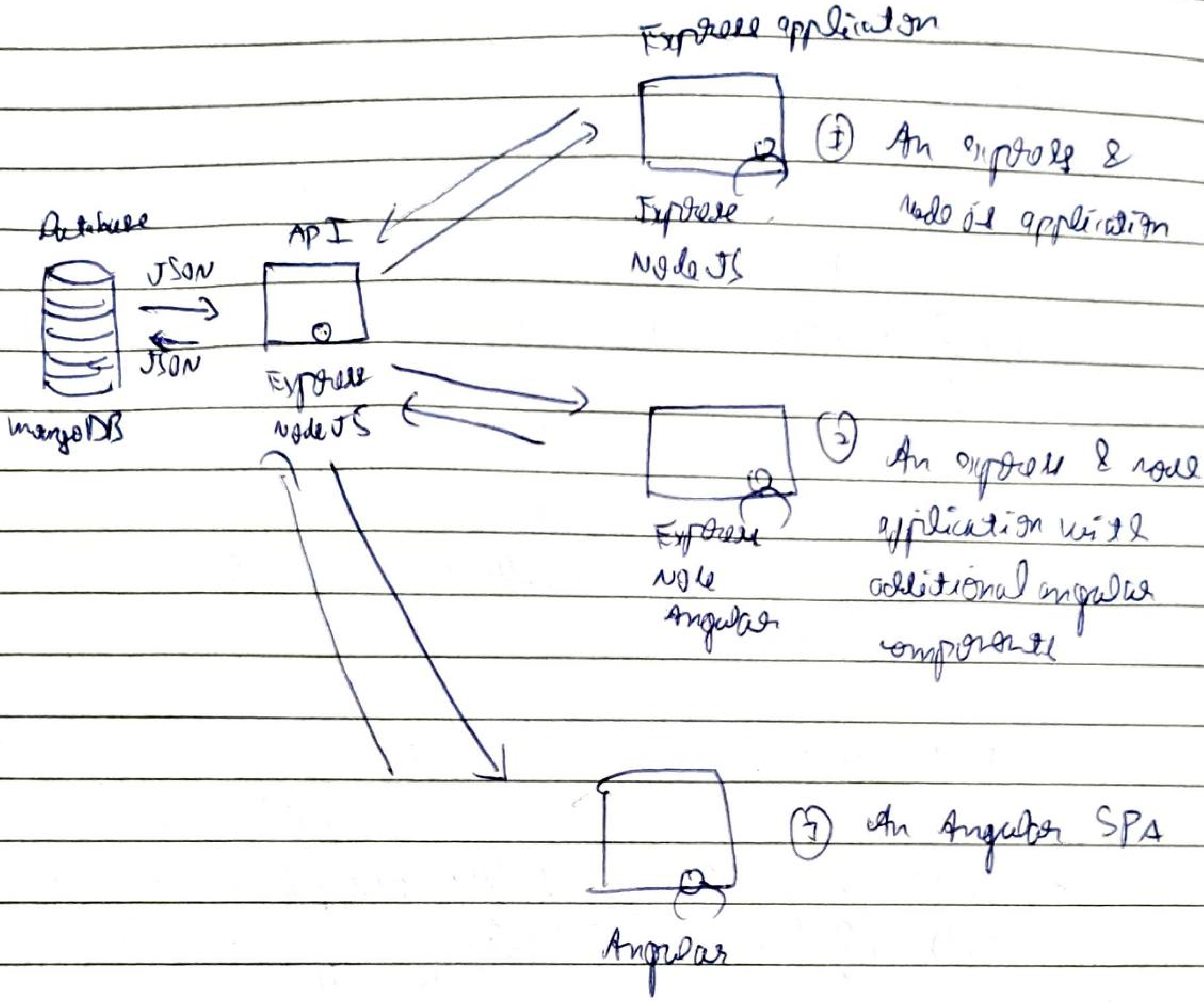
- **Hosting with Heroku ->**

- Heroku is a hosting platform which have several settings tweaked for Node JS deployment.

- Applications on Heroku are essentially git repos, making the publishing process incredibly simple.

Q MEAN stack architecture?

=>



(3) The
3 ways of architecture

PAGE :
DATE: / /

Unit 2 Building an Express & Node.js web application ->

Q Installing Node.JS on different OS ?

=> Installing on Windows ->

(1) Installation can be done using standalone installer

(2) navigate to 'nodejs.org/download/' page.

(3) Download right .msi file for 32-bit or 64-bit version.

(4) Run the installer after downloading it.

(5) Click on next & installation will begin

(6) Once installation is finished, click on finish to exit.

Installing on Mac OS ->

(1) Same as windows

Installing on Linux ->

(1) Installation is done using tarball file

(2) navigate to 'nodejs.org/download/' page & download suitable 'tar.gz' file

(3) Then through make command, install node.js.

Q Intro to NPM?

⇒ • Node JS uses module system that allows to extend the platform.

• NPM is a registry of packages to browse, download & install third-party modules

• It is also a CLI tool used to manage local & global packages

• The best way to install, update & remove nodeJS modules is to use NPM.

• <u>Installing a package using NPM</u> ->

- npm install <package name>
- npm install -g <package name>

• <u>removing a package</u> ->

- npm uninstall <package name>
- npm uninstall -g <package name>

• <u>updating a package</u> ->

- npm update <package name>
- npm update -g <package name>

PAGE :
DATE: / /

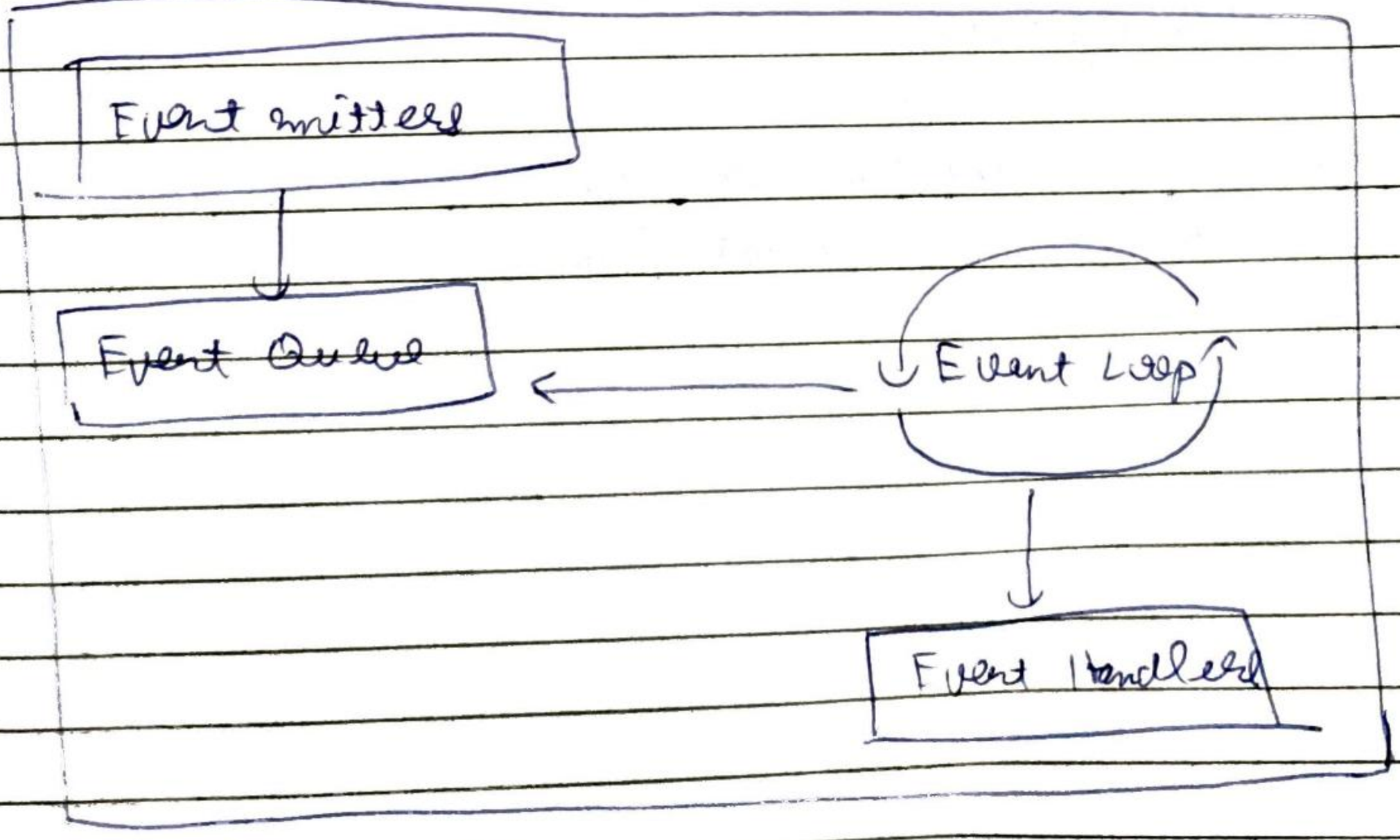
Q Javascript event driven programming.

=) • Javascript is an event-driven language, which means that you register call to specific events, & that code will be executed once the event is emitted.

• This concept allows to execute logic based code without blocking the rest of the program from running.

• The browser manages a single thread to run the entire Javascript code using an inner loop, commonly referred to as the event loop.

• Every time an event is emitted, the browser adds it to an event queue. The loop will then grab the next event from the queue in order to execute the event handler registered to that event.



PAGE :
DATE: / /

Q Node JS event driven programming?

⇒ • When developing web server logic, a lot of system resources are wasted on blocking code.

• To solve this issue many web platforms have implemented a thread pool system that usually issues a single thread per connection.

• This has its own disadvantages such as →
(i) managing threads becomes a complex task
(ii) system resources are wasted on idle threads
(iii) scaling these kinds of applications cannot be done easily.

• Using event driven architecture will help dramatically reduce the load on the server while implementing Javascript's asynchronous behaviour in building your web application.

• This approach is made possible thanks to a simple design pattern, which is called observer.

PAGE :
DATE: / /

Q. What is JavaScript Closure?

=> • Closures are functions that refer to variables from their parent environment.

• Ex code ->

```
function parent() {
    var message = "Hello World";

    function child() {
        alert(message);
    }

    return child;
}

var childFN = parent();
childFN();
```

• As from the code above, we can see that the variable childFN is function parent() in it, & not only the function, but its lexical environment (all the reference of variables) is also passed in the variable.

• So we can call the childFN() at any time of the program & it will have access to its lexical environment variable message.

• This process is known as JavaScript Closure.

PAGE :
DATE: / /

Q Different node modules?

⇒ The common JS standards specify the following three key components when working with modules →

- require(): This method is used to load the module into your code.
- exports: This object is contained in each module & allows you to expose pieces of your code when the module is loaded.
- module: This object is used to provide metadata info about the module.

- There are different types of modules →

(1) Node JS core modules →

- Node JS core modules come preloaded with Node & are documented in great detail in its documentation.

- The core modules provide most of the basic functionalities of Node, including file system access, HTTP & HTTPS interfaces, & much more.

- To load a core module, you just need to use the require method in JavaScript file.

- Ex fs = require('fs');

PAGE :
DATE: / /

(5) Node JS third party module →

- NPM installs these modules in a folder named node_modules under the root folder of application.

- To use a third-party module, you can just require them as you would normally require a core module.

- Node will first look for the module in the core modules folder & then try to load the module from node_modules folder.

- Ex: const express = require('express');

(3) Node JS file modules →

- We can also place the modules inside a folder & load them by providing the folder path.

- Ex: var hello = require('./modules/hello');

- Node will look for modules folder & inside that, look for hello module & require it.

(4) Node JS folder modules →

- Requiring folder modules is done in same way as file modules

var hello = require('./modules/hello');

PAGE :
DATE: / /

- Now if we folder named module exists, Node will go looking for package.json. In package.json, it will looks for "main" object & try to open that module.

- If package.json file doesn't exist or main property is not defined, Node will automatically try to load the index.js file in the folder.

(8) Developing Node.js web application.

=>

(8) Installing express

=> - NPM can be used to install express on your project.

- Write the command → npm install express

- You can also create a package.json file with the its content like

```
{
  "name": "MEAN",
  "version": "0.0.3",
  "dependencies": {
    "express": "~4.8.8"
  }
}
```

- Now write the command →
  npm install

PAGE :
DATE: / /

- How express will be installed.

Q Create first express app?

⇒ • First create a package.json file in your directory.

• Then create server.js file with following code in it →

```
var express = require ('express');
var app = express();

app.use('/', function (req, res) {
    res.send ('Hello World.');
});

app.listen (3000);
console.log ("Server running on port 3000");
```

• Now in the directory, execute the command →

node server.js

• Now the application is running on →
http://localhost:3000

PAGE :
DATE: / /

Q Request & Response Objects?

Requst →

=> . The request provides a handful of helpful methods that contain the information needed about current HTTP request.

- req.query : This is an object containing the parsed query string parameters.
- req.params : This is an object ~~used~~ containing the parsed routing parameters
- req.body : This is an object used to retrieve the parsed request body.
- req.param(name) : This is used to retrieve a value of a request parameter.
- req.ip : This is used to retrieve current remote IP.

Response →

- Any request sent to the server will be handled & responded using the response object methods.
- res.status(code) : This is used to set the response HTTP status code
- res.cookie(name, value) : This is used to set a response cookie

PAGE :
DATE: / /

- res.redirect ([status], url) : This is used to redirect the request to a given URL.

- res.send ([body], [body]): This is used for non-streaming responses.

- res.json ([body], [key]): This is used to send an object or an array.

Q- Horizontal folder structure?

=>. A horizontal project structure is based on the division of folders & files by their functional role rather than by the features they implement.

- This means that there is a single controller or folder that contains all of the application controllers, a single models folder that contains all of the application models & so on.

- Ex:->

app
- controllers
- models
- routes
- views

config
- env
- config.js
- express.js

PAGE :
DATE: / /

public
- config
- controllers
- CSS
- directives
- filters
- img
- services
- views
- application-js

server.js
package.json

- The app folder is where all the express application logic is.

- The config folder is where you keep your express application configuration files.

- The public folder is where you keep your static assets files files

- The package-json file is the metadata file that helps you to organise your application dependencies.

- The server.js file is the main file of Node JS application.

PAGE :
DATE: / /

Q Vertical folder structure ?

=> A vertical folder structure is based on the division of folders & files by the feature they implement

- Core
  - client
  - server
- features
  - client
  - server
- server.js
- package.json

- The server folder
- The core folder contains the main application files.
- The features folder include the additional features -
- The server folder is where you keep your features server logic.
- The client folder is where you keep your feature's client-side files.

PAGE :
DATE: / /

Q File naming convention?

=>.

- Feature.js (problem)
- Locate

=>. Client
- config
  - feature.js
- controller
  - feature.js

Server
- config
  - feature-js
- controller
  - feature-js
  :

- So many files with same name bcoz MVC structure is followed.

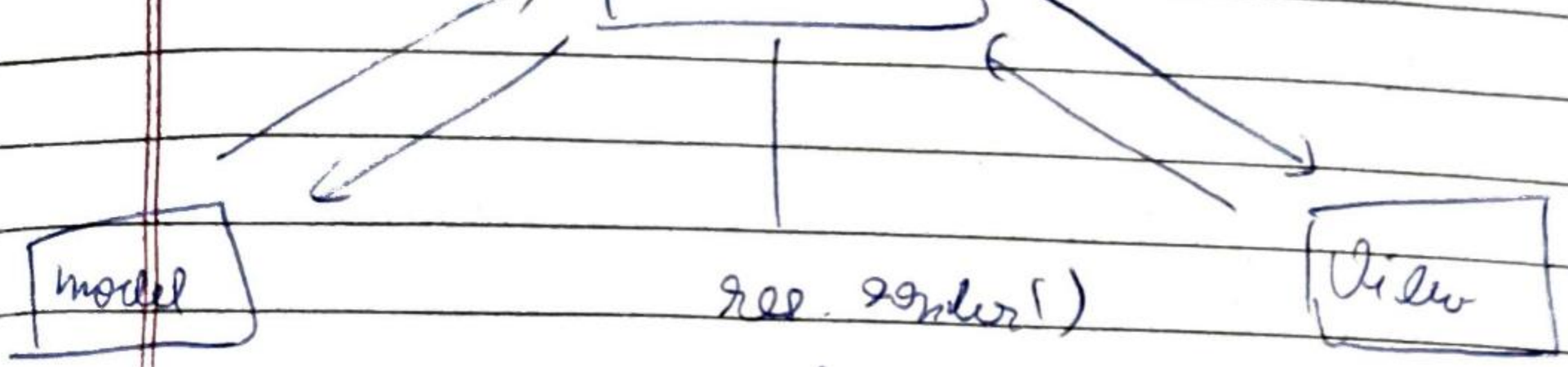
Set
- Separate different name for config & controllers.
  Ex. feature.config-js , feature.controller.js

- But MVC is applied parallely for Express & Angular.
  ∴ best solution
  feature.server.config.js, feature.client.config-js

PAGE :
DATE: / /

Q Rendering Views?

>>



↓

HTML output

- The basic concept of rendering view is passing your data to a template engine that will render the final view usually in HTML.

- Express has two methods for rendering views: ~~app.render()~~ ~~which is~~

  - app.render(): used to render the view & then pass the HTML to callback function

  - res.render(): renders the view locally & sends the HTML as a response

PAGE :
DATE: / /

Q Serving static files?

⇒ • express.static() middleware

• use it as well →

app.use(express.static('./public'));

• express.static() takes one argument to determine the location of the static folder.

Unit 3 Angular JS & mongoDB →

Q Key features of mongoDB?

⇒ ① It is BSON format →

• mongoDB stores data in the BSON (Binary encoded JavaScript Object Notation) format.

• It is designed to be more efficient in size & speed.

• BSON documents are a simple data structure representation of objects as they are in key-value format.

• BSON uses the -id field as primary key which will be a unique identifier.

PAGE :
DATE: / /

(2) MongoDB queries :-

(2) MongoDB indexing :-

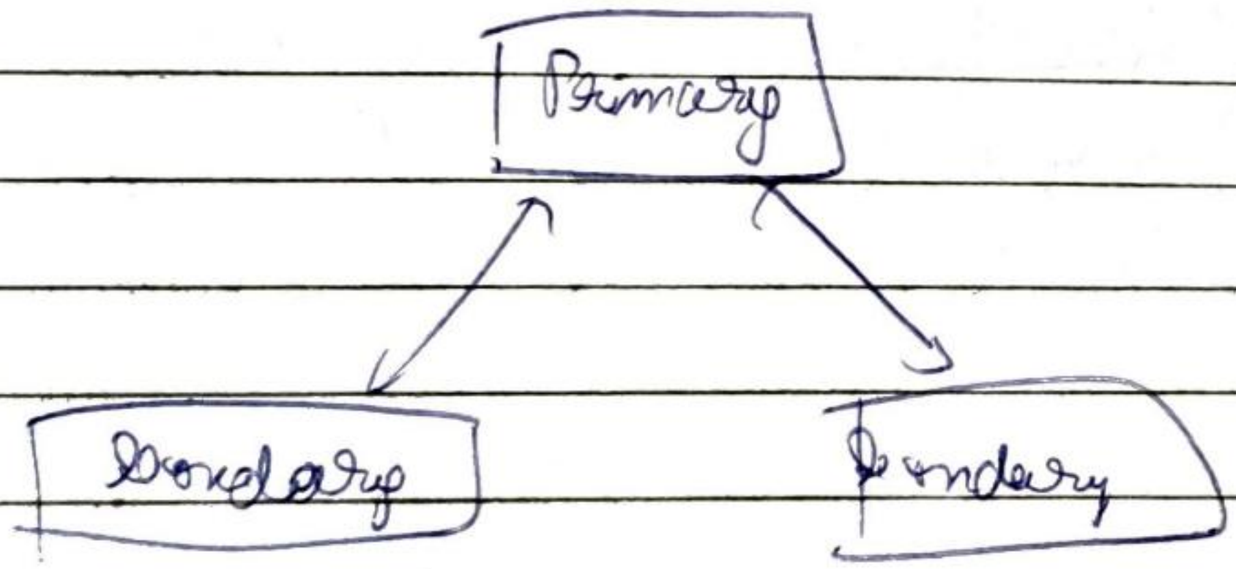
- Indexes are unique data structure that enable database engine to efficiently resolve queries.

- mongoDB uses a predefined index which can speed up the scan for data

(3) MongoDB replica set :-

- To provide data redundancy & improved availability mongoDB uses an architecture called replica set

- Replication of datasets helps protect your data to recover from hardware failure & increase read capacity.



(3) MongoDB Sharding :-

- Sharding is the process of splitting the data b/w different machines or shards

- Each shard holds a portion of the data & functions as a

separate database.

- The collection of several clients together is what forms a single logical database.

Q What is mongoDB shell?

=> - To interact with mongoDB, mongoDB shell is used -

- The mongoDB shell is a command line tool that enables the execution of different operations using a JavaScript syntax query language.

PAGE :
DATE: / /

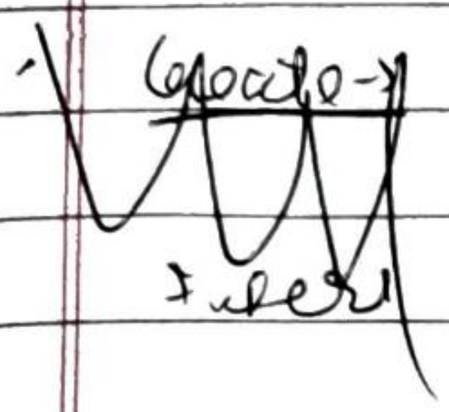
Q MongoDB CRUD operations ?

=> CRUD → Create Read Update Delete

- Firstly create a new database with command →

  use web-development

db with name web-development created & using



- now create a new collection
  db.createCollection ('courses')

- create →

  Insert one document →

```
db.courses.insertOne ({
  "name": "Ultimate React",
  "price": 50
})
```

Insert multiple documents →

~~db.courses.insertMany~~

~~db.insert~~

db.courses.insert(

{

"name": "Ultimate React",

"price": 80

},

{

"name": "complete path to JSX",

"price": 80

}

)

- Read →

Find one document →

db.courses.findOne()

Find multiple documents →

db.courses.find()

Find documents by value →

db.courses.find({

"name": "Ultimate React"

})

PAGE :
DATE: / /

- Update :

Update one document ->

```
db.courses.updateOne(
  { "_id": 1 },
  { $set:
    { "name": "Ultimate ReactJs" }
  }
)
```

Update multiple ->

```
db.courses.update(
  { "price": 50 },
  { $set: { "price": 499 } }
)
```

Increment field value ->

```
db.courses.update(
  { "_id": 2 },
  { $inc: { "views": 1093 } }
)
```

Rename field ->

```
db.courses.updateOne(
  { "_id": 3 },
  { $rename:
    { "views": "likes" }
  }
)
```

PAGE :
DATE: / /

- Delete →

To delete a document →

db.courses.remove(
  { "name": "Ultimate React.js" }
)

Q Understanding mongoose schema ?

⇒. Mongoose use a schema object to define the document list of properties, it will use own type & constraints, to enforce the document structure.

- Once specifying a schema, you will go on to define a model constructor that you will use to create instances of mongoDB documents.

Q Mongoose model methods ?

⇒. Birth

- Defining a schema →

```
const userSchema = new mongoose.Schema ({
  name : {
    first : String,
    last : String
  },
  age : { type : Number },
})
```

PAGE :
DATE: / /

Defining a model ->

const User = mongoose.model ('User', userSchema);

User u = new User ({

name : {

first : 'Sid',

last : 'Empty'

},

age : 21

})

Q mongoose model methods?

=> . find () : find document

- count() : returns a count
- findById () : return a single document by I.D
- findById And Update () : executes find & then update by ID
- findById And Remove () :
- findOne () : return a single document
- findOneAndUpdate () : executes
- create () : create document object & save it to db
- remove () : remove document

PAGE :
DATE: / /

Q Angular JS modules?

=> • With angular JS everything is encapsulated in modules.

• Application module :-

• Every Angular JS application needs atleast one module to bootstrap, it is referred as application module.

• It is created & retrieved with command :
angular.module(name, [requires], [configFn])
- name : String of module name
- requires : Array of strings defining other modules dependencies
- configFn : Fn that will run when module is registered

• External modules :-

• To support the continuous development of the framework, Angular JS team breaks Angular's functionality into external modules.

• These modules are being developed by the same team that works on core framework

• Third Party modules :-

• In the same way the Angular JS team supports its external modules, it also encourages outside vendors to create third party modules.

• It extends the framework functionality & provides

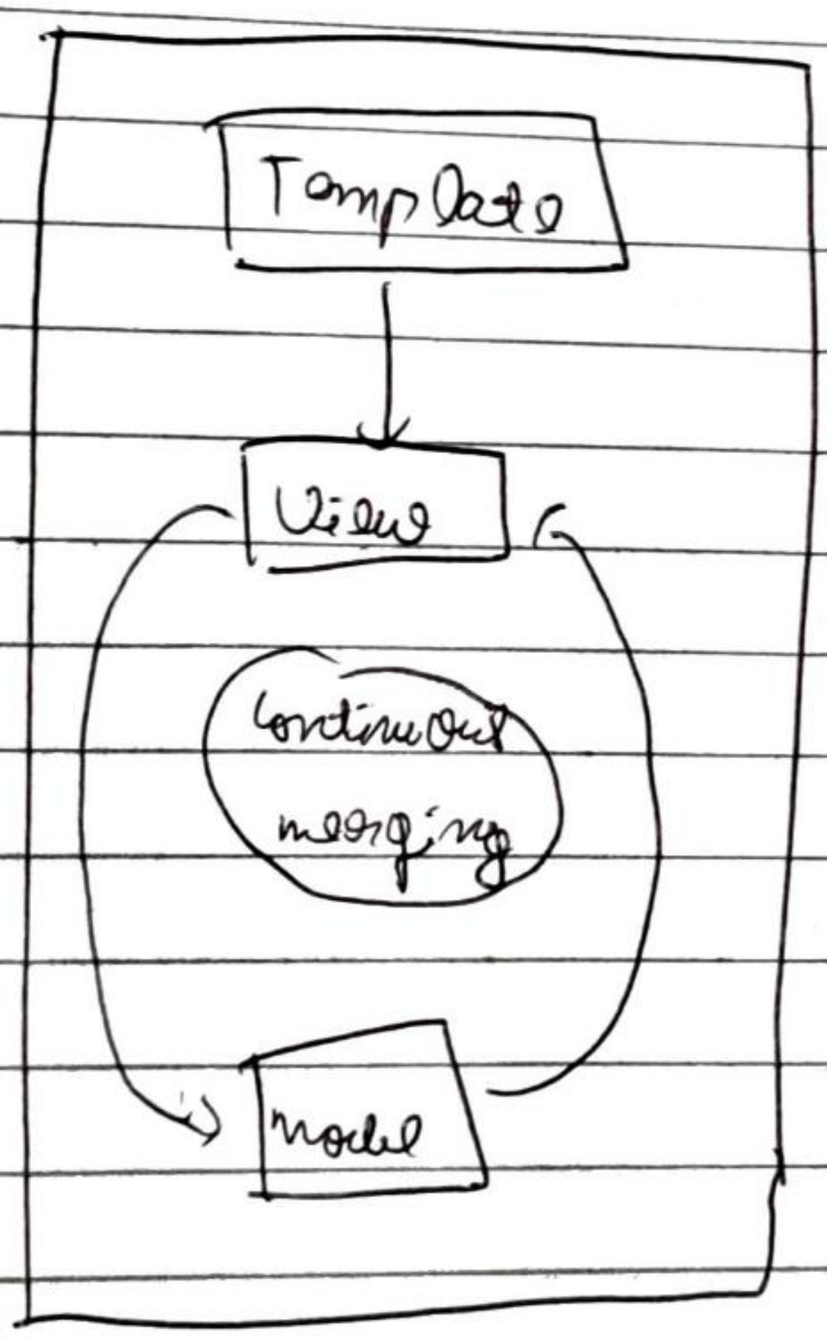
developers with an easier starting point.

Q Two way data binding in Angular JS?

→ ~ # Two-way binding makes Angular to applications to always keep the model synchronized with view & vice versa.

- Angular JS uses the browser to compile HTML templates, which contain special directives & binding instructions that produce a live view.

- Any events that happen in the view automatically update the model, while any changes occurring in the model immediately get propagated to the view.



PAGE :
DATE: / /

Q What is Directives?

⇒. Directives are markers, usually attributes or elements, which enable the AngularJS compiler to attach a specified behaviour to a DOM element or its child.

Basically directives are the way AngularJS interacts with DOM elements & are what enabled the basic operation of an AngularJS application.

- Core directives →

- AngularJS comes prebundled with necessary directives, which define the functionality of an Angular application.

- Ex ng-app : placed on DOM element you want to use as the root application

ng-controller : This tells the compiler which controller to use

ng-model : This is placed on input elements

ng-show / ng-hide : This shows & hides an element according to boolean expression

PAGE :
DATE: / /

- Custom directives

- The user can write its own custom directives in AngularJS.

- Custom directives keep the application cleaner & more readable.

Q Bootstrapping ~~any~~ AngularJS application?

=>

Q Install ing AngularJS ?

=> (1) Bower dependencies manager :-

- Bower is a package manager tool designed to download & maintain frontend third party libraries.

- Bower is a Nodejs module so to begin using it, you will have to install it globally using npm

npm install -g bower

(2) Configuring Bower dependencies manager :-

- Bower installation process downloads the packages content & automatically place them under a bower_components default folder.

- .bowerrc file is used.

PAGE :
DATE: / /

③ Installing AngularJS using bower →

- Go to bower.json file & add the line →

```
dependencies: {
    angular: ~1.2
}
```

- now write following command →

```
bower install
```

- This will install angularJS

Q Structuring Angular JS application ?

=>

```
main
  - config
  - controllers
      - main.client.controller.js
  - CSS

section1
  - config
  - controllers
      - section1.client.controller.js

section2
  - configjs
  - controllers
      - section2.client.controller.js
```

- Each file is placed in a proper folder with a proper file name that usually describe what sort of code it contains.

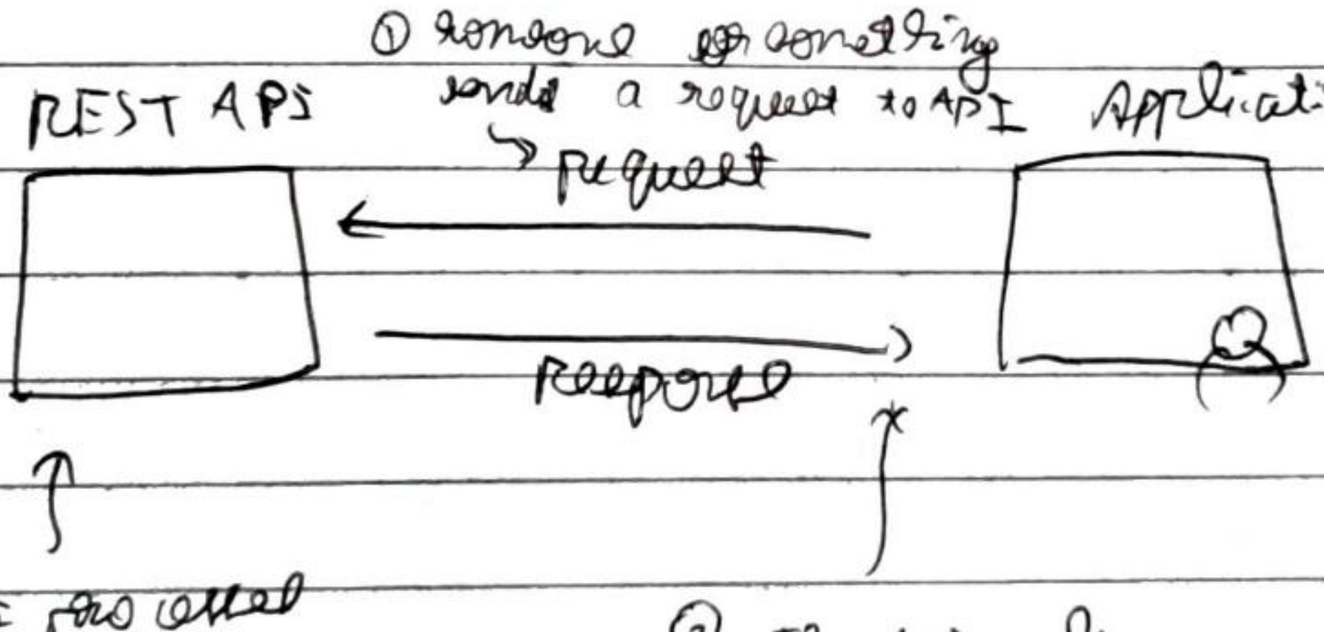
- It is a horizontal structure where entities are arranged in modules & folders according to their type & a main application file is placed at the root folder of the application.

Q ~~Explain~~

Unit 4 Advanced Topics →

Q Rules for writing a REST API ?

⇒ • REST API is used to create a stateless interface to your database, enabling a way for other applications to work with the data



PAGE :
DATE: / /

① Request URLs →

- Within URLs, some actions generally have to be carried like
  - create a new item
  - read a list of several items
  - read a specific item
  - update a specific
  - delete a specific item

- Ex
  Create new location → / locations
  Read list of locations → / locations
  etc

② Request methods →

- HTTP requests have different methods with it that eventually tell the server what type of action to take.

- Each HTTP request performs CRUD operations.

method	use
POST	Create a new data in the db
GET	Read data from db
PUT	Update a document in db
DELETE	Delete an object from the db

PAGE :
DATE: / /

③ Responses & Status codes →

- Every single API request should return a response.
- There are two key components to a response :
  - The returned data
  - The HTTP status code

- Returning data from an API →

- The API should return a consistent data format.
- API will return one of these three :
  - A JSON object containing data answering the query
  - A JSON object containing error data
  - A null response

- Using HTTP status codes →

- A good RFST API should return the correct HTTP status code

- Some examples :
  - 200 - OK
  - 201 - Created
  - 404 - Not found
  - 500 - Internal server error

Q Setting up the API in express?

=> ① Creating the router ->

- Including the router in application ->

const indexRouter = require ('./routes/index');

app.use('/', indexRouter);

- Specifying the request methods in routes ->

router.get('/location', () => {fn()});

- Specifying required URL parameters ->

/api/locations/:locationId

② Creating controller placeholders ->

const locationsCreate = (req, res) => { };

③ Returning JSON from express to client ->

res.status

const locationsCreate = (req, res) => {

res

- status (200)
- json ({"status": "success"});

PAGE :
DATE : / /

(4) Indexing the model ->

const loc = mongoose.model('location');

(5) Testing the API ->

- Testing the API through postman

Q methods to read, add, update & delete data in mongoDB in REST API

=> (1) GET method : Reading data from DB ->

Action	method	URL Path
Read a list of locations	GET	/locations
Read a specific location	GET	/locations/:locationid
Read a specific review	GET	/locations/:locationid /reviews/:reviewid

- Finding a single document in mongoDB using mongoose ->

~~findById~~

findById();

(2) POST method : Adding data to MongoDB ->

Action	method	URL path
Create new location	POST	/locations
Create new review	POST	/locations/:locationid/ reviews

• Creating new documents in mongo DB using mongoose :)

create (data to save, callback);

(3) PUT method : Updating data in mongoDB →

Action	method	URL path
Update a specific location	PUT	/locations/:locationid
Update a specific review	PUT	/locations/:locationid/reviews/:reviewid

• Updating the document in mongo DB using mongoose →

findByIdAndUpdate (id, data, callback);

(4) DELETE method : Deleting data from mongoDB →

Action	method	URL path
Delete a specific location	DELETE	/locations/:locationid
Delete a specific review	DELETE	/locations/:locationid/reviews/:reviewid

• Deleting documents in mongo DB using mongoose →

findByIdAndRemove (id, callback);

PAGE :
DATE: / /

Q How to call an API from express?

=>. Not only our own API, we can call any API from express.

① Adding the request module ->

- Request module helps to make a request to any other API.

- Install it with command ->

```
npm install request
```

- Now require the module ->

```
const request = require('request');
```

② Using the request module ->

- ```request(options, callback);```

options: Object defining the request

callback: Function to run when a response is received

PAGE :
DATE: / /

- Four most common options :-

Option	Description
url	Full URL of the request
method	method of the request
json	Body of the req as a JS obj.
qs	JS obj representing parameters

- ~~const requestObject =~~

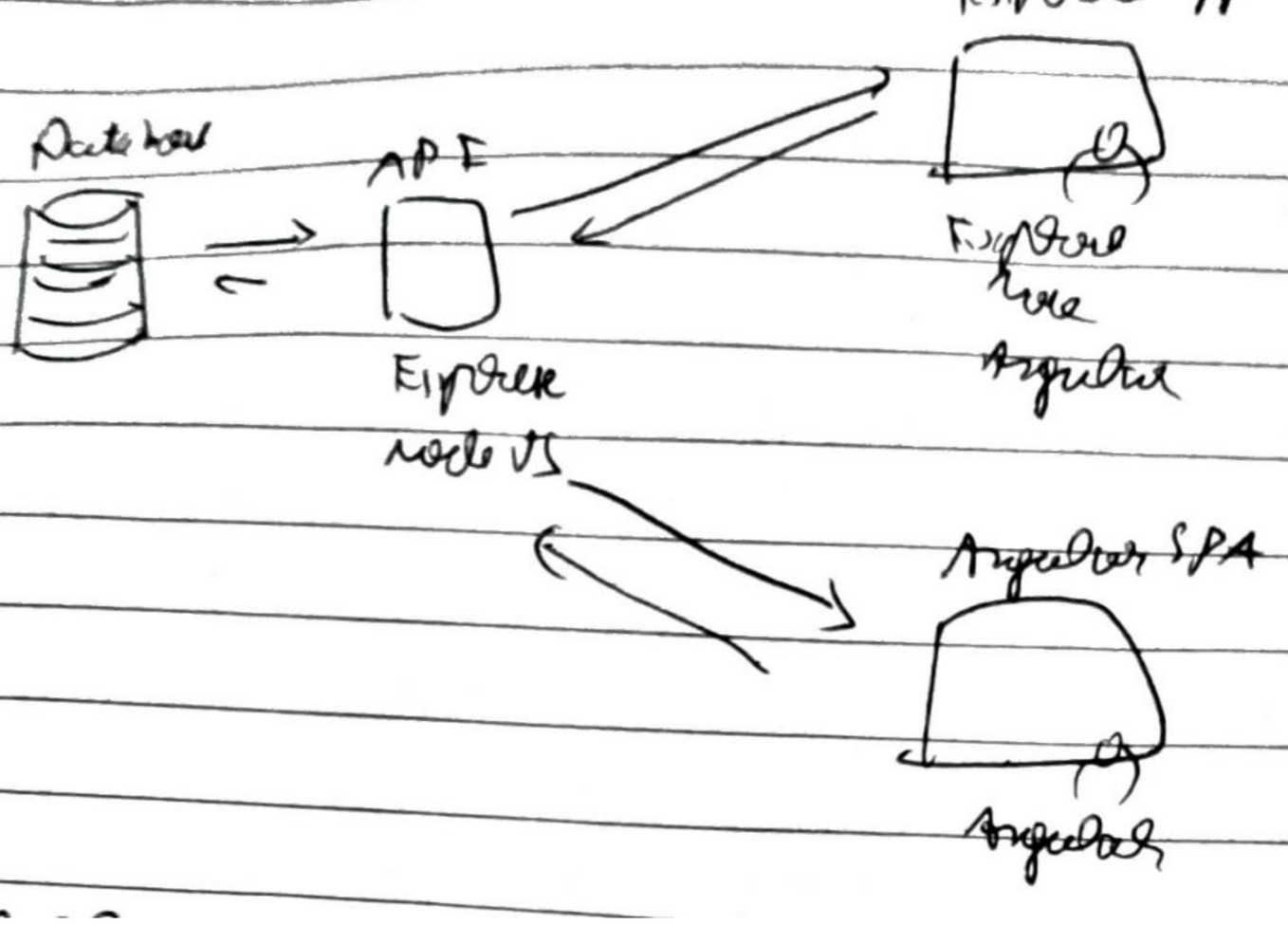
- const requestOptions = {
    url: 'http://yourapi.com/api/path';
    method: 'GET',
    json: { }
  };

- (err, response, body) => {
    if (err) {
      console.log (err);
    } else if (response.statusCode === 200) {
      console.log (body);
    } else {
      console.log (response.statusCode);
    }
  }

PAGE :
DATE: / /

Q Angular application with IS diagram?

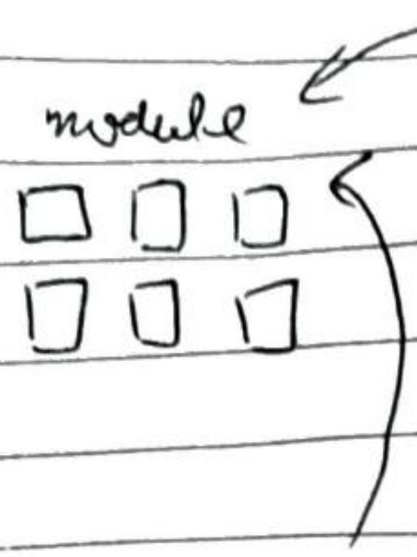
=>

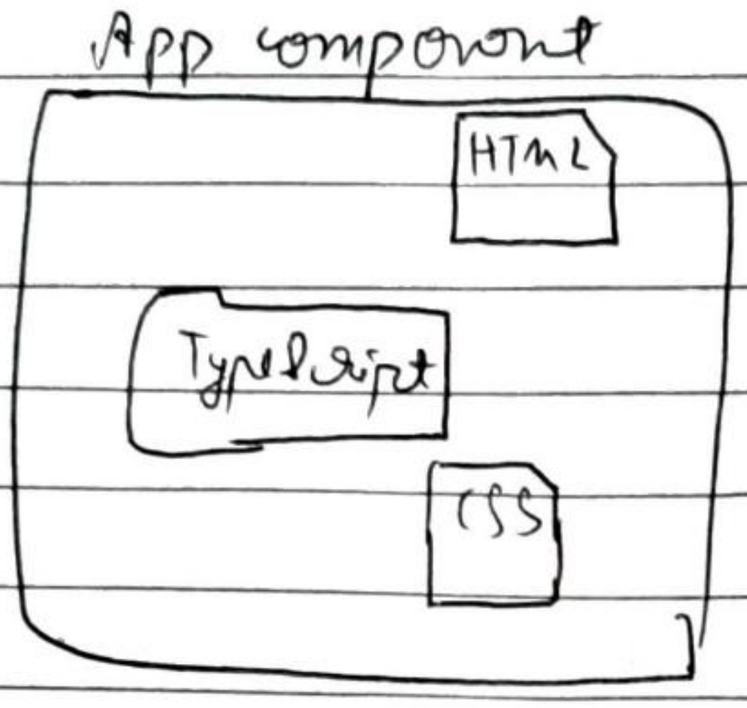


PAGE :
DATE: / /

Q Working with Angular Components?

=> • module native Angular Components





- The component App component comprise three files: Typescript, HTML & CSS

- The Typescript file is the key part of the component, defining the functionality referencing the other files & declaring which selector (HTML tag) it will bind to.

- The component Typescript file exports the AppComponent class

- The module file imports the AppComponent class from the component Typescript file & declares it as the entry point into the application.

- The module file also imports various pieces of native Angular functionality.

PAGE :
DATE: / /

- Fx component →

① Create a new home-list component →

ng generate component home-list

note

② make it default component, open app.module.ts →

bootstrap : [HomeListComponent]

③ Creating HTML template →

- Start with creating some static HTML with hard coded data

④ moving data out of the template into the code →

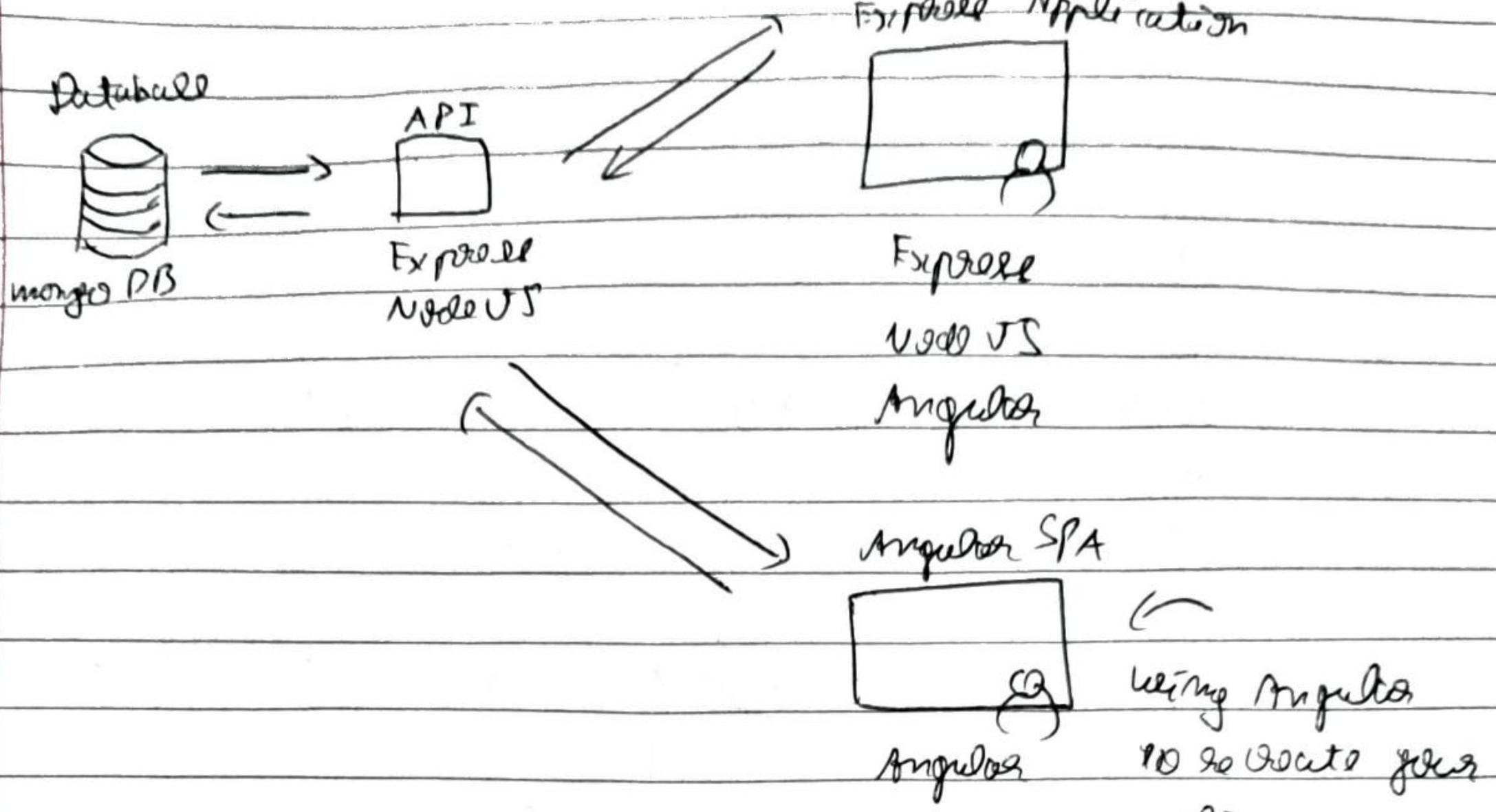
- With Angular, you can define a class member inside component code & bind it to HTML by using { }

⑤ Using class member data in the HTML template

PAGE :
DATE: / /

Q Building a simple page application with Angular ?

=>



① Adding navigation in an Angular SPA →

- Add the routings to your application with →
  @angular / router

② Building a modular app using multiple nested components →

- The homepage has three main sections
  - Page header
  - List of locations
  - Side bar

PAGE :
DATE: / /

(3) Safely binding HTML content →

- Angular lets you pass through some HTML tags if properly binding it used instead of default bindings.

(4) Routing parameters →

- Your routes for the app are defined in app.module.ts

- Parameters can also be added to the routes.

- Ex path: 'location/:locationId'

(5) Working with forms & handling submitted data →

- Inline form action can be added to add data to form →

<form action="" >

- By adding new number to data service to POST new review to API, we can submit data to API. →

.post(url, formData)

PAGE :
DATE: / /

Unit 3 some questions ans →

Q Working of AngularJS routing?

⇒. AngularJS team developed the 'ngRoute' module that allows you to define URL paths & their corresponding templates.

- ngRoute will manage the routing activity in the browser

- TO mark a URL, hashbang are used as it helps in routing as well as SEO optimization.

~~Ex localhost:3000/#!/~~

Ex localhost:3000/#!/example

- $routeProvider provides several methods to define routing behaviour.

Ex $routeProvider.
when('/', {
templateURL: 'views/example.html'
})

PAGE :
DATE: / /

Q Working of Angular JS service?

=> Angular JS service are used to share information between different entities of the same Angular JS application.

- Service can be used to fetch data from your server, share cached data, & inject global objects into other components.

- Angular JS provided services ->

  - $http: used to handle AJAX requests
  - $resource: used to handle RESTful APIs
  - $location: used to handle URL manipulations

  etc.

- Creating Angular JS services ->

Creating services can be done by using three ->

  - provider(): It provides way to define a service
  - service(): Initiate a new object from service
  - factory(): Provide the value returning from service function