**Express JS**

* Most popular Node js framework
* Super fast response light
* Unopinionated, minimalist framework
* Server side framework
* For Best REST API backend
* It's too easy to manage any backend server
* **Install express**   
  npm install express
* **Create an express application**   
  let express = require(‘express’);  
  let app = express();
* **Create a basic express server**   
  app.get(‘/’, ( req, res ) => {  
   res.send(‘hello world’);  
  });  
    
  app.listen(port , () => {  
   console.log(‘server is running on port ’);  
  });

**Express Basic Routing**

* To create a standard REST API we have to manage route for this
* Get, post, put, patch & Delete route operation is the main concern for API
* **GET** route with express   
  app.get(‘/users’, ( req, res ) => {  
   res.send(‘GET Route is Done’);  
  });
* **POST** route with express   
  app.post(‘/users’, ( req, res ) => {  
   res.send(POST Route is Done’);  
  });
* **PUT / PATH** route with express   
  app.put(‘/users/:id’, ( req, res ) => {  
   res.send(PUT / PATCH Route is Done’);  
  });
* **DELETE** route with express   
  app.delete(‘/users/:id’, ( req, res ) => {  
   res.send(PUT / PATCH Route is Done’);  
  });
* Get url params value by params   
  app.delete(‘/users/:id’, ( req, res ) => {  
   let delete\_id = req.params.id;   
  });

**Express Responses**

app.delete(‘/users’, ( req, res ) => {  
 res.send();  
 res.sendFile();  
 res.json();

res.status();   
 res.render();  
 res.redirect()  
});

**Static Folder**

let path = require(‘path’);  
 app.use(express.static(path.join(\_\_dirname, ‘public’)));   
 app.use( ‘/images/’ express.static(path.join(\_\_dirname, ‘public/images’)));   
app.use( ‘/css/’ express.static(path.join(\_\_dirname, ‘public/css)));

**Setup html project**

* To setup a html project in express server first create a project folder like public
* Then create html pages
* Now load any pages by routing   
  app.get(‘/about’, ( req, res ) => {  
   res.sendFile(‘public/file.html’, { root : \_\_dirname });  
  });

**Express Server Structure**

/controller  
/models  
/middleware  
/routes   
/config   
/views  
server.js

**Express Router**

* To decorate all routes in one place
* Make a routes folder for all routes
* Now create a route file like student.js
* Init express router   
  let express = require(‘express’);  
  let router = express.Router();
* Now make all route by using this router   
  router.get(‘/users’, ( req, res ) => {  
   res.send(‘GET Route is Done’);  
  });  
  router.post(‘/users’, ( req, res ) => {  
   res.send(POST Route is Done’);   
  });
* Now exports router from router
* After export now import router and use it on server   
  app.use(‘/api/students’, require(‘file path of router’));
* Some combined route   
  router.route(‘/’).get( controllerName ).post( controllerName );

**Body Data with POST**

* Init body parser middleware for get body data   
  app.use(express.json());  
  app.use(express.urlencoded({ extended : false }));

app.post(‘/users’, ( req, res ) => {  
 res.send(req.body);  
 });

**Express Middleware**

* Middleware is a function which is run when a req is fired and between a response send
* We can filter all route by middleware
* Middleware function structure   
  let logger = ( req, res, next ) => {  
   console.log(‘Logger middleware id fired’);  
   next();   
  }
* Now use middleware on global area   
  app.use(logger);
* Use middleware on route handle   
  app.get(‘/’, middleware, ( req, res, next ) => {  
   res.send(‘hello world’);  
  });

**Express Controller**

* Controllers are the functions of touting operation
* Just create a folder with all controllers file
* Exports all function to route

**Express Multer**

* Multer is a express helper package for manage form data with file
* We will use it for manage any kind of app file
* Its a express middleware with form data parser
* First install multer   
  npm install multer
* Now init it to your server   
  const upload = multer({  
   dest : ‘./media/students/’,   
   fileFilter : (req, file, cp) => {  
   cb(null, true);  
   },   
   limits : 1024,   
     
  });
* storage setup   
  const storage = multer.diskStorage({  
   destination : (req, file, cb) => {  
    
   },  
   filename : (req, file, cb) => {  
    
   }  
  });
* now use upload methods for file management   
  upload.single(‘file\_name’);  
  upload.array(‘file\_name’, 10);  
  upload.none();  
  upload.fields([  
   { name : ‘file\_name’ , maxCount : 5 },  
   { name : ‘file\_name’ , maxCount : 5 }  
  ]);
* Get file info with all details   
  req.file

**Express handlebars**

* Express handlebars are template engine
* We can build an complete apps by using this
* First install express handlebars   
  npm install express-handlebars
* Now init handlebars   
  const exphbs = require(‘express-handlebars’);  
    
  app.engine(‘handlebars’, exphbs.engine());  
  app.set(“view engine”, “handlebars”);   
  app.set(“views”, “./views”);
* Now create you main layout files in views/layouts   
  main.handlebars
* Create a structure of template   
  {{{ body }}}
* Now render an view   
  app.get(‘/homepage’, (req, res) => res.render(‘view name’))
* Pass data to view   
  app.get(‘/homepage’, (req, res) => res.render(‘view name’, {  
   name : data ,  
  }))
* Get data value  
  {{ keyName }}
* Some syntax of handlebars   
  {{#each students }}  
   {{ this.name }}

{{/each}}  
  
 {{#if students }}  
 {{ this.name }}

{{/if}}

**Errors Handle by Middlewares**

* To manage error custom way we can use errors hander
* Its like a middleware with error
* First create a error handler   
  app.use( (error, req, res, next) => {  
   const errorStatus = err.status || 500;  
   const errorMessage = err.status || 500;  
   return res.status().json({  
   success : false,   
   status : errorStatus,   
   message : errorMessage,   
   Stack : error.stack  
   });   
    
  } );
* **Create error controller**   
  const createError = ( status, message) => {  
   const error = new Error();  
   error.status = status  
   error.message = message;   
   return error;  
  }

Cors