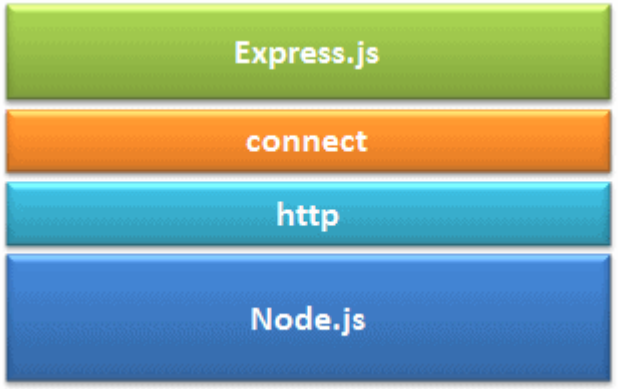
Express.js

What is Express.js

- Express.js is a free and open source web application framework for node.js
- It is used for designing and building the web application easily and quickly.
- Web applications are web apps that can run on a web browser.
- It is light weight and helps to organize the web application on the server side into more organized MVC architecture.

- Express.js is based on the Node.js middleware module called connect which in turn uses http module.
- So, any middleware which is based on connect will also work with Express.js.



Features

1. Faster server side development:

- Express.js provides may commonly used features of node.js in the form of functions that can be readily used anywhere in the program.
- This removes the need to code for several hours and thus saves time.

2. Middleware

- Middleware is a part of the program that has access to the database, client request and the other middleware.
- It is mainly responsible for the systematic organization of different functions of express.js

3. Routing

- ExpressJs provides a highly advanced routing mechanism which helps to preserve the state of the web page with the help of their URLs.

4. Templating:

- ExpressJS provides template engine that allow developers to build dynamic content on the web pages by building HTML templates on the server side.

5. Debugging:

- Debugging is crucial for the successful development of web applications.
- ExpressJS makes debugging easier by providing a debugging mechanism that has ability to pinpoint the exact part of the web application which has bugs.

Advantages of Express.js

1. Makes Node.js web application development fast and easy.
2. Easy to configure and customize.
3. Allows you to define routes of your application based on HTTP methods and URLs.
4. Includes various middleware modules which you can use to perform additional tasks on request and response.
5. Easy to integrate with different template engines like Jade, Vash, EJS etc.
6. Allows you to define an error handling middleware.
7. Easy to serve static files and resources of your application.
8. Allows you to create REST API server.
9. Easy to connect with databases such as MongoDB, MySQL
10. A web developer can use Javascript as a single language for both front end and backend development. The developer does not need to learn or any other language for server side development.

Installing Express.js

- To install Express.js first you need to create a project directory and create a package.json file which will be holding the project dependencies.
- npm install –g express
- This command will install latest version of express.js globally on your machine so that every node.js application on your machine can use it.
- npm install express --save
- This command will install latest version of express.js local to your project folder.

Starting Express server

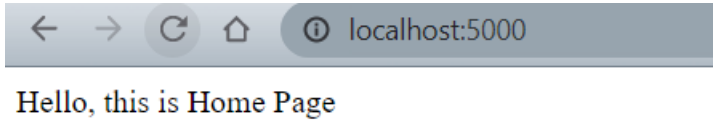
- Express server is where you will handle logic to integrate your routes and HTTP request methods.
- The request argument contains information about the GET request, while response.send() dispatches data to the browser.
- The data within response.send() can be a string, object, or an array.

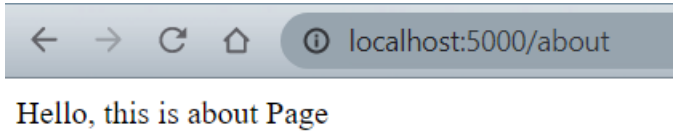
Express.js Fundamental Concepts: Routing and HTTP Methods

- Routing is used to determine the specific behavior of the application.
- Each route can contain more than one handler functions, which is executed when the user browses for a specific route.
- app.method(path, handler)
- app: app is an instance of express.js
- Method is an HTTP method such as get,put, delete
- Path is the route to the server for a specific web page
- Handler is the callback function that is executed when the matching route is found.

```
const express=require('express');
var app= express()
app.get('',(req,res)=>{
res.send('Hello, this is Home Page');
})
app.get('/about',(req,res)=>{
  res.send('Hello, this is about Page');
  })

  app.listen(5000)
```





CRUD Operations

- Create- create new resource
- Read – Read resource from server
- Update – Update resource
- Delete – Delete a resource

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

app.get('/', function (req, res) {
   res.send('<html><body><h1>Hello World</h1></body></html>');
});

app.post('/submit-data', function (req, res) {
   res.send('POST Request');
});

app.put('/update-data', function (req, res) {
   res.send('PUT Request');
});

app.delete('/delete-data', function (req, res) {
   res.send('DELETE Request');
});
 var server = app.listen(5000, function () {
   console.log('Node server is running..');
});
```

Routing:

- Rout is a section of Express code that associates as HTTP verb(GET,POST,PUT,DELETE), a URL path/pattern and a function that is called to handle that pattern.

Middleware:

- Middleware functions are the functions that have the access to the request object(req) and response object(res) and the next middleware function in the application request response cycle.
- These functions are used to modify req and res object for the tasks like parsing the request bodies/adding responses headers etc.
- Order of method is extremely important.

Middleware functions can perform the following tasks:

- Execute any code.
- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware in the stack.

If the current middleware function does not end the request-response cycle, it must call `next()` to pass control to the next middleware function.

Otherwise, the request will be left hanging.

To load the middleware function, call `app.use()`, specifying the middleware function.

```
const express = require('express')
const app = express()

const Logging = function (req, res, next) {
  console.log('LOGGED')
  next()
}

app.use(Logging)
app.get('/', (req, res) => {
  res.send('Hello World!')
})

 app.listen(3000)
```

Every time the app receives a request, it prints the message “LOGGED” to the terminal.

The order of middleware loading is important: middleware functions that are loaded first are also executed first.

If `Logging` is loaded after the route to the root path, the request never reaches it and the app doesn’t print “LOGGED”, because the route handler of the root path terminates the request-response cycle.

The middleware function `Logging` simply prints a message, then passes on the request to the next middleware function in the stack by calling the `next()` function.

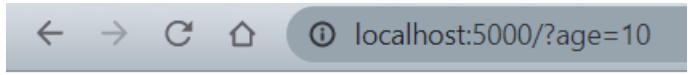
```
const express = require('express')
const app = express()

const requestTime = function (req, res, next) {
  req.requestTime = Date.now()
  next()
}
app.use(requestTime)
app.get('/', (req, res) => {
  let responseText = 'Hello World!<br>'
  responseText += `<small>Requested at: ${req.requestTime}</small>`
  res.send(responseText)
})
 app.listen(3000)
```

```
const { response } = require('express');
const express=require('express');
var app= express();
const reqfilter=(req,res,next)=>{
  if(!req.query.age)
  {
    res.send("Please provide age")
  }
  else{
next();
}}
app.use(reqfilter)
app.get('',(req,res)=>{
res.send('Hello, this is Home Page');
})
app.get('/user',(req,res)=>{
  res.send('Hello, this is User Page');
  })

  app.listen(5000)
```

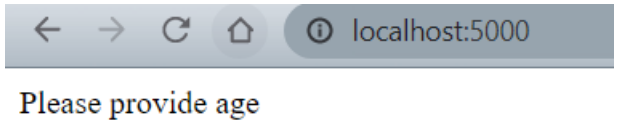
Please provide age

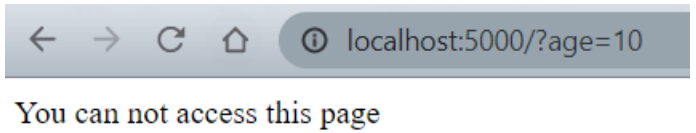


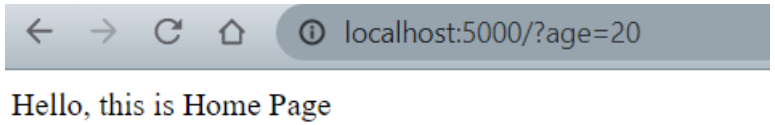
Hello, this is Home Page

```
const { response } = require('express');
const express=require('express');
var app= express();
const reqfilter=(req,res,next)=>{
  if(!req.query.age)
  {
    res.send("Please provide age")
  }
  else if (req.query.age<18){
    res.send("You can not access this page")
  }
  else{
next();
}}
app.use(reqfilter)
app.get('',(req,res)=>{
res.send('Hello, this is Home Page');
})
app.get('/user',(req,res)=>{
  res.send('Hello, this is User Page');
  })

  app.listen(5000)
```







Apply Middleware on a single Route

```
const { response } = require('express');
const express=require('express');
var app= express();
const reqfilter=(req,res,next)=>{
  if(!req.query.age)
  {
    res.send("Please provide age")
  }
  else if (req.query.age<18){
    res.send("You can not access this page")
  }
  else{
next();
}}
```

```
//app.use(reqfilter)
app.get('',(req,res)=>{
res.send('Hello, this is Home Page');
})
app.get('/user',reqfilter,(req,res)=>{
 res.send('Hello, this is User Page');
 })
 app.get('/about',(req,res)=>{
  res.send('Hello, this is About Page');
 })
 app.listen(5000)
```

Express Generator:

- Quickly create an application skeleton
- Easily get standard application shell for quick and rapid prototyping

What is MongoDB?

- MongoDB is document database with the scalability and flexibility that you want with the querying and indexing that you need.
- A document oriented database provides APIS or query/update language that exposes the ability to query or update based on internal structure in the document.
- MongoDB documents are composed of pairs and have the following structure
- {

  Field1:valuea,

  Field2:value2

  }

- MongoDB is NoSQL database.
- Data is stored in the form of collections
- Collections doesn’t have rows and columns

Connect to MongoDB

- Var mongoose=require(‘mongoose’);
- Create properties file
- Import the properties file in index.js
- Mongoose.connect(dbURL)
- Mongoose.connection.on(‘connected’,()=>{
- Console.log(‘connected’);
- }