Node JS

AS AN ASYNCHRONOUS EVENT DRIVEN JAVASCRIPT RUNTIME, NODE IS DESIGNED TO BUILD SCALABLE NETWORK APPLICATIONS.

NodeJs VS Tomcat

	Node JS/JS	Tomcat/Java
Solidez		✓
Ubicuidad	✓	
Mejores IDEs		✓
Proceso Build/Deployment	✓	
Debug Remoto		✓
Queries a BD	✓	
Librerías		✓
Manejo de JSON	✓	
Hilos		✓
Rapidez	✓	

Arquitectura de Aplicación Tradicional

Request/Response Model Processing Steps:

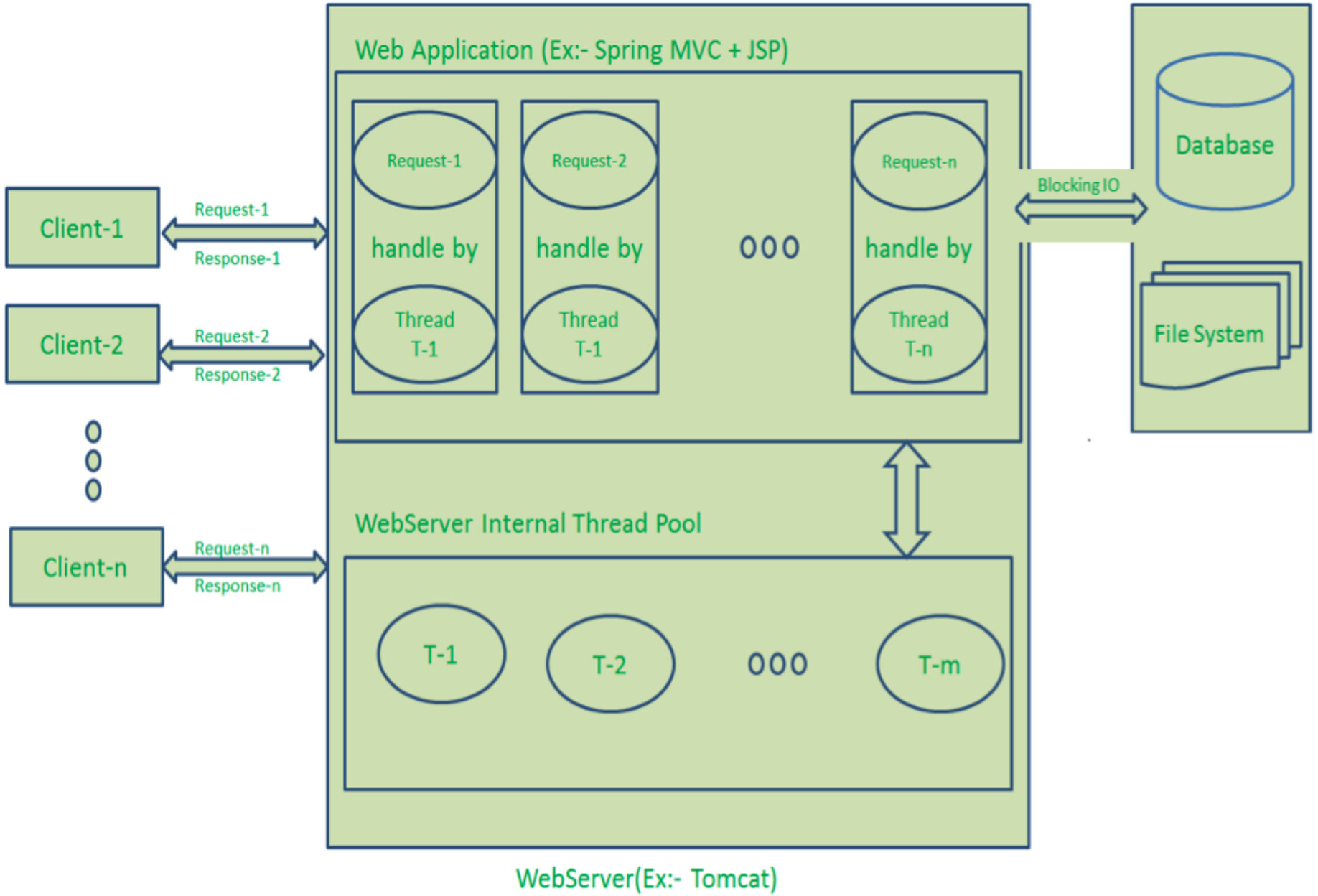
Clients send request to Web Server.

Web Server internally maintains a Limited Thread pool to provide services to the Client Requests.

Web Server is in infinite Loop and waiting for Client Incoming Requests

Web Server receives those requests.

- Web Server pickup one Client Request
- Pickup one Thread from Thread pool
- Assign this Thread to Client Request
- This Thread will take care of reading Client request, processing Client request, performing any Blocking IO Operations (if required) and preparing Response
- This Thread sends prepared response back to the Web Server
- Web Server in-turn sends this response to the respective Client.


Web Application (Ex:- Spring MVC + JSP)
Request-1
Request-2
Request-n
handle by
handle by
000
handle by
Thread T-1
Thread T-1
Thread T-n
Client-1
Request-1
Response-1
Client-2
Request-2
Response-2
Client-n
Request-n
Response-n
Blocking IO
Database
File System
WebServer Internal Thread Pool
T-1
T-2
000
T-m
WebServer(Ex:- Tomcat)

Arquitectura Node JS

Single Threaded Event Loop Model Processing Steps:

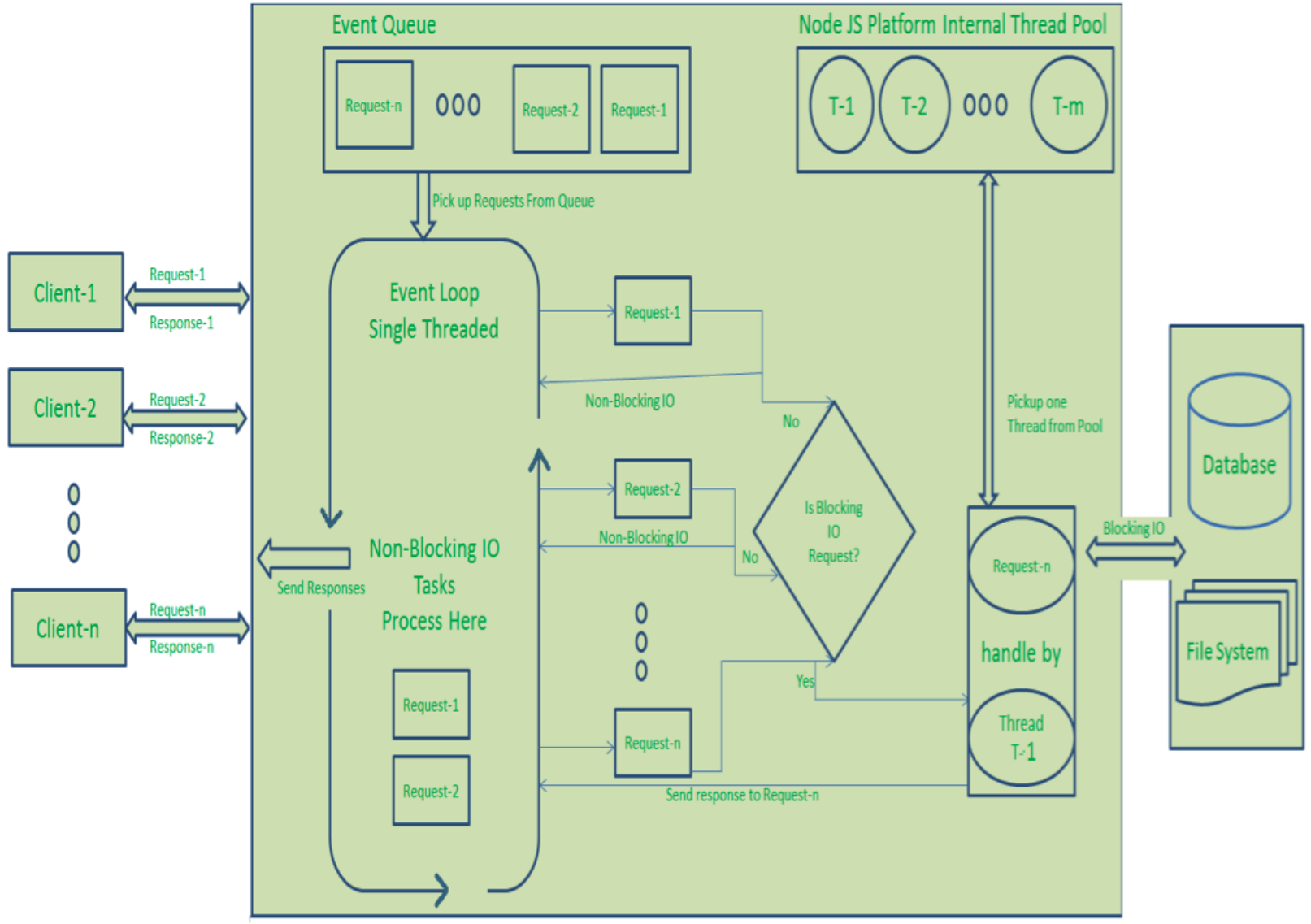
Clients Send request to Web Server.
Node JS Web Server internally maintains a Limited Thread pool to provide services to the Client Requests.
Node JS Web Server receives those requests and places them into a Queue. It is known as “Event Queue”.
Node JS Web Server internally has a Component, known as “Event Loop”. Why it got this name is that it uses indefinite loop to receive requests and process them. (See some Java Pseudo code to understand this below).

Event Loop uses Single Thread only. It is main heart of Node JS Platform Processing Model.

Even Loop checks any Client Request is placed in Event Queue. If no, then wait for incoming requests for indefinitely.

If yes, then pick up one Client Request from Event Queue

- Starts process that Client Request
- If that Client Request Does Not requires any Blocking IO Operations, then process everything, prepare response and send it back to client.
- If that Client Request requires some Blocking IO Operations like interacting with Database, File System, External Services then it will follow different approach
  - Checks Threads availability from Internal Thread Pool
  - Picks up one Thread and assign this Client Request to that thread.
  - That Thread is responsible for taking that request, process it, perform Blocking IO operations, prepare response and send it back to the Event Loop
  - Event Loop in turn, sends that Response to the respective Client.



Node JS Application/Node JS Server

Ventajas de Arquitectura Node JS

1) Handling more and more concurrent client's request is very easy.

2) Even though our Node JS Application receives more and more Concurrent client requests, there is no need of creating more and more threads, because of Event loop.

3) Node JS application uses less Threads so that it can utilize only less resources or memory

Hello World!

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

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Módulos

Module	Descritpion
assert	Provides a set of assertion tests
buffer	To handle binary data
child_process	To run a child process
cluster	To split a single Node process into multiple processes
crypto	To handle OpenSSL cryptographic functions
dgram	Provides implementation of UDP datagram sockets
dns	To do DNS lookups and name resolution functions
domain	Deprecated. To handle unhandled errors
events	To handle events
fs	To handle the file system
http	To make Node.js act as an HTTP server
https	To make Node.js act as an HTTPS server.
net	To create servers and clients
os	Provides information about the operation system
path	To handle file paths
punycode	Deprecated. A character encoding scheme
querystring	To handle URL query strings
readline	To handle readable streams one line at the time
stream	To handle streaming data
string_decoder	To decode buffer objects into strings
timers	To execute a function after a given number of milliseconds
tls	To implement TLS and SSL protocols
tty	Provides classes used by a text terminal
url	To parse URL strings
util	To access utility functions

Query Params en Node

*/**end-point***?*id*=8888&*param*=x

```
1   var http = require('http');
2   var url = require('url');
3
4   http.createServer(function (req, res) {
5       res.writeHead(200, {'Content-Type': 'text/html'});
6       var queryParams = url.parse(req.url, true).query;
7       var txt = "id: " + queryParams.id + "\nparam: " + queryParams.param;
8       res.write(txt);
9       res.end();
10  }).listen(8080);
```

Manejo de archivos

Módulo ‘fs’

Dinámica con NodeMailer

Instalar nodemailer con npm:

The Nodemailer module can be downloaded and installed using npm:

```
C:\Users\Your Name>npm install nodemailer
```

Dinámica con NodeMailer (2)

```
var nodemailer = require('nodemailer');

var transporter = nodemailer.createTransport({
  service: 'hotmail.com',
  auth: {
    user: 'fulanitor@hotmail.com',
    pass: 'xxxxxxx'
  }
});

var mailOptions = {
  from: 'fulanito@hotmail.com',
  to: 'fulanito@iteso.mx',
  subject: 'Enviando e-mail con Node.js',
  text: 'Qué facil!'
};
```

```
transporter.sendMail(mailOptions, function(error, info){
  if (error) {
    console.log(error);
  } else {
    console.log('E-mail enviado: ' + info.response);
  }
});
```

REST con Node JS

Installing

Assuming you've already installed Node.js, create a directory to hold your application, and make that your working directory.

```
$ mkdir myapp
$ cd myapp
```

Use the `npm init` command to create a `package.json` file for your application. For more information on how `package.json` works, see Specifics of npm's package.json handling.

```
$ npm init
```

This command prompts you for a number of things, such as the name and version of your application. For now, you can simply hit RETURN to accept the defaults for most of them, with the following exception:

```
entry point: (index.js)
```

Enter `app.js`, or whatever you want the name of the main file to be. If you want it to be `index.js`, hit RETURN to accept the suggested default file name.

Now install Express in the `myapp` directory and save it in the dependencies list. For example:

```
$ npm install express --save
```

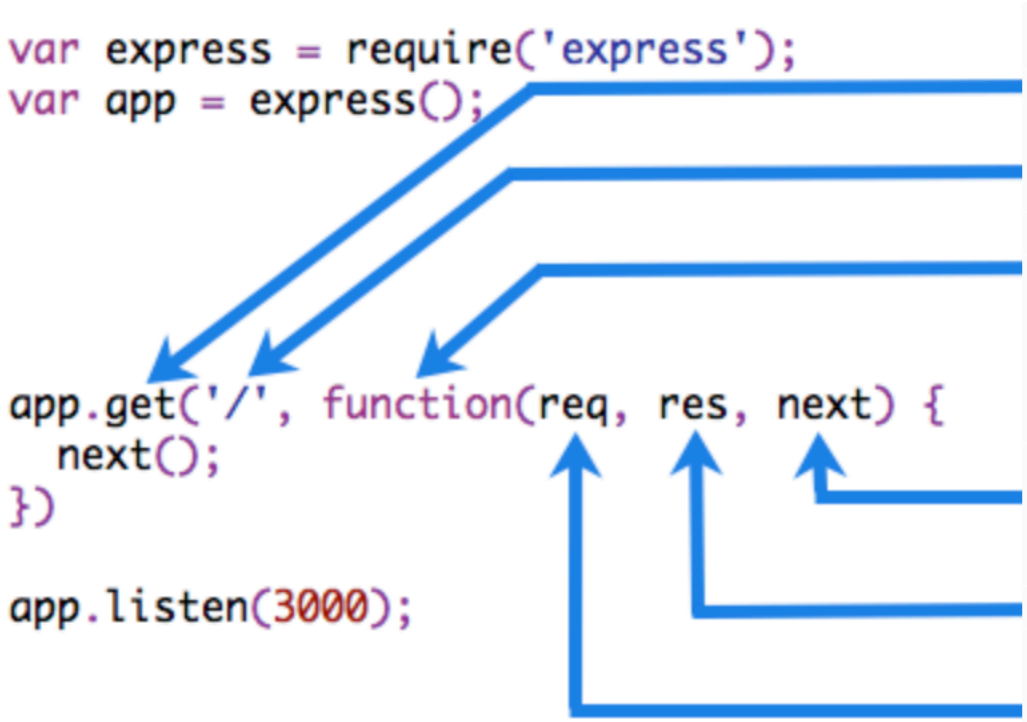
To install Express temporarily and not add it to the dependencies list:

```
$ npm install express --no-save
```

Hello World con Express JS

```
1  const express = require('express' 4.16.2 )
2  const app = express()
3
4  app.get('/', function (req, res) {
5    res.send('Hello World!')
6  })
7
8  app.listen(3000, function () {
9    console.log('Example app listening on port 3000!')
10 })
```

Análisis de un http call en Express JS



HTTP method for which the middleware function applies.

Path (route) for which the middleware function applies.

The middleware function.

Callback argument to the middleware function, called "next" by convention.

HTTP response argument to the middleware function, called "res" by convention.

HTTP request argument to the middleware function, called "req" by convention.

Equivalente al @PathParam en Express JS

```
// handler for the /user/:id path, which renders a special page
router.get('/user/:id', function (req, res, next) {
  console.log(req.params.id)
  res.render('special')
})
```

Equivalente al @HeaderParam en Express JS

```
// predicate the router with a check and bail out when needed
router.use(function (req, res, next) {
  if (!req.headers['x-auth']) return next('router')
  next()
})
```