

NodeJS

Background

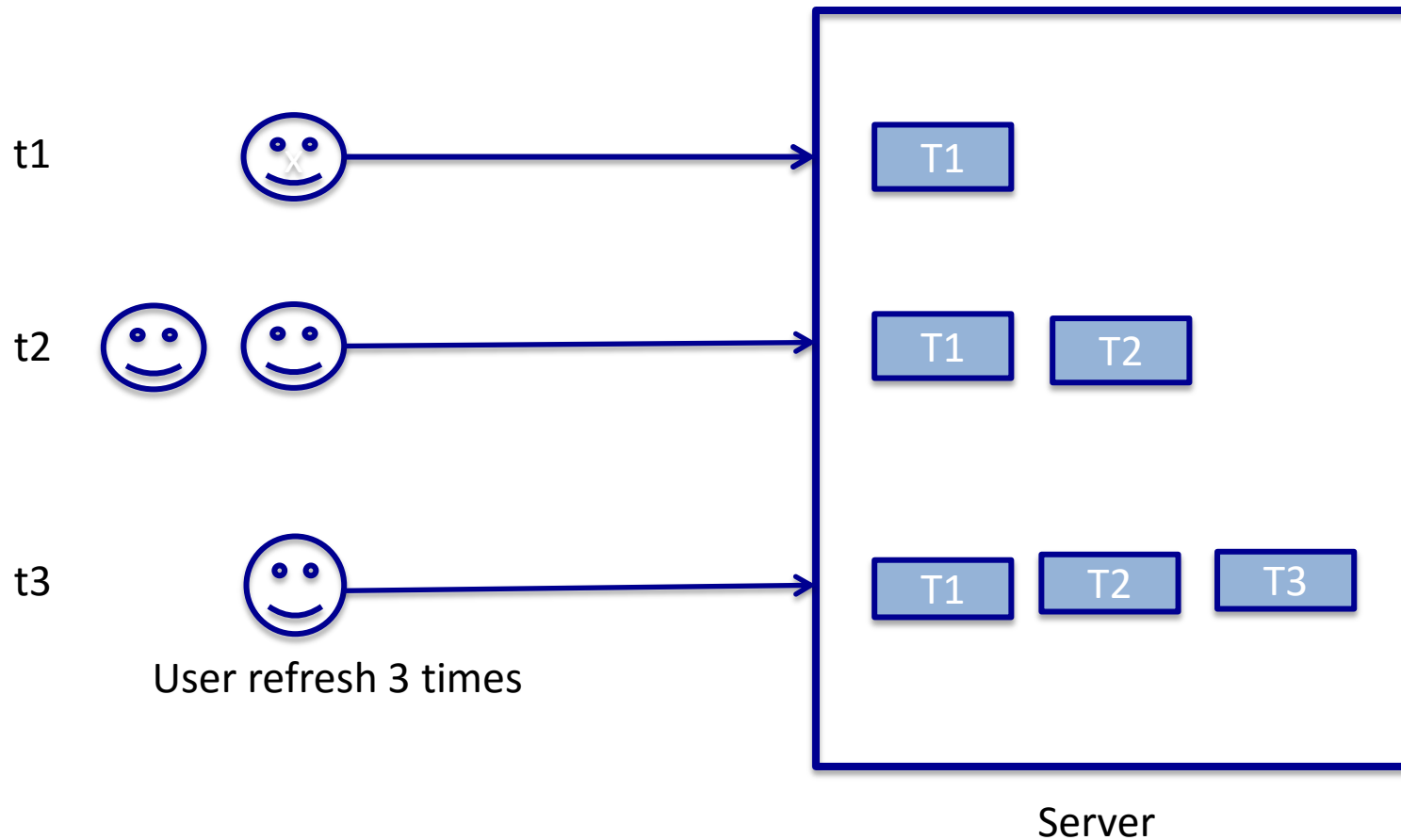
- Node.js runs on V8.
- V8 is an open source JavaScript engine developed by Google. Its written in C++ and is used in Google Chrome Browser.
- It was created by Ryan Dahl in 2009.
- Is Open Source. It runs well on Linux systems, can also run on Windows systems.
- Latest version: v4.0.0

Evolution of web

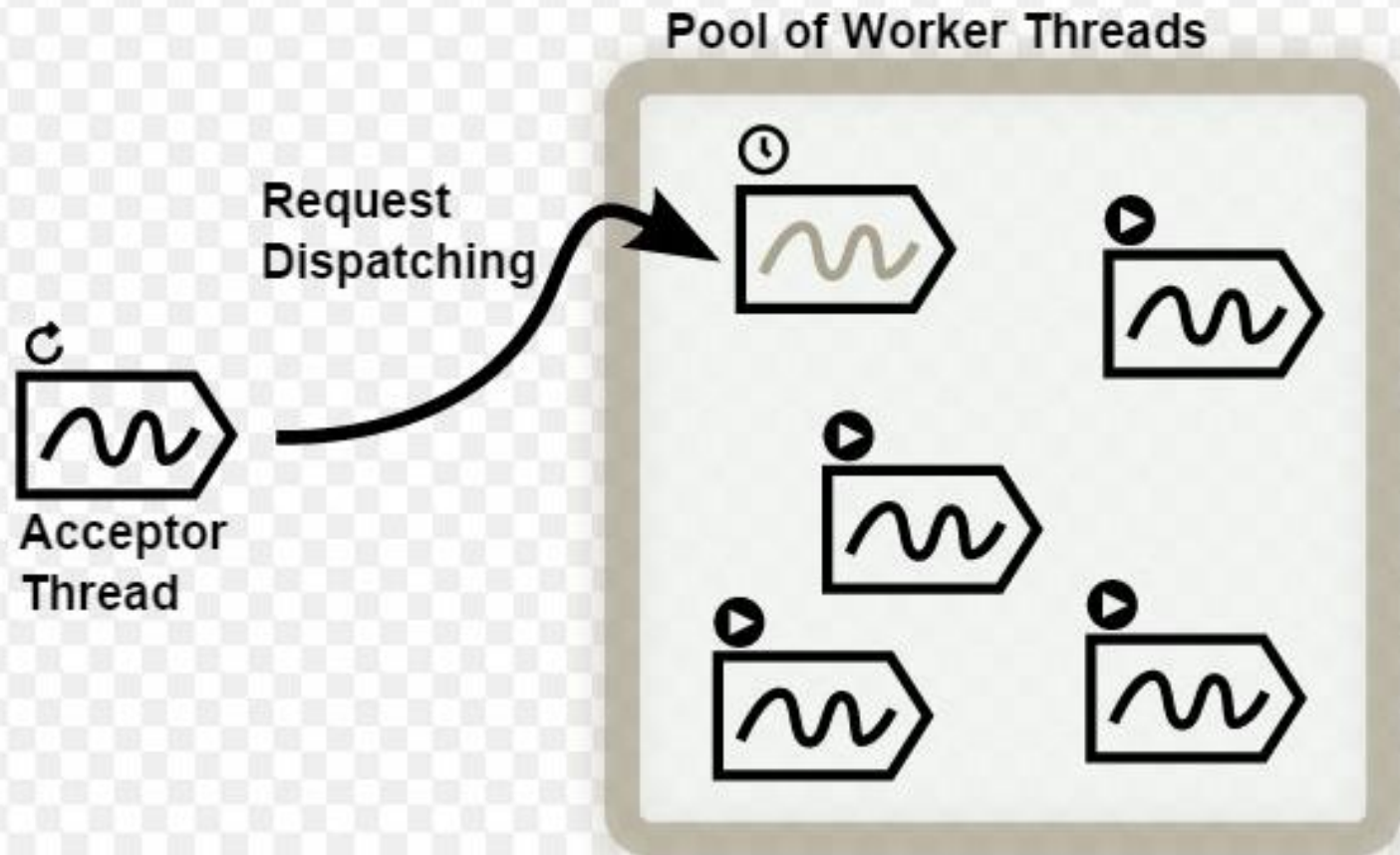
Web has evolved from

- Static websites (90's)
- Dynamic web applications (AJAX) (early 2000's)
- Real time web applications (Notifications)

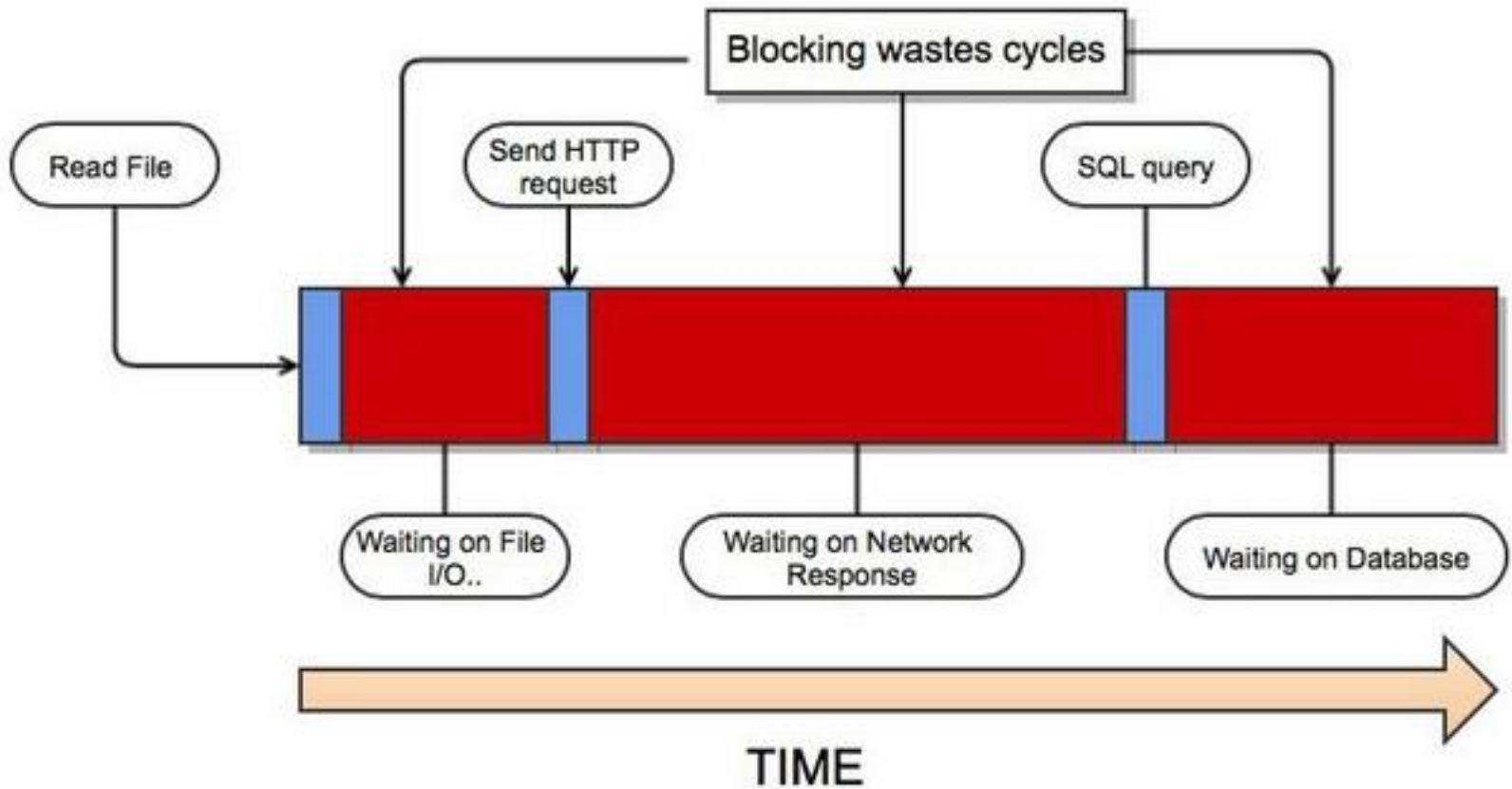
Traditional multi threaded server



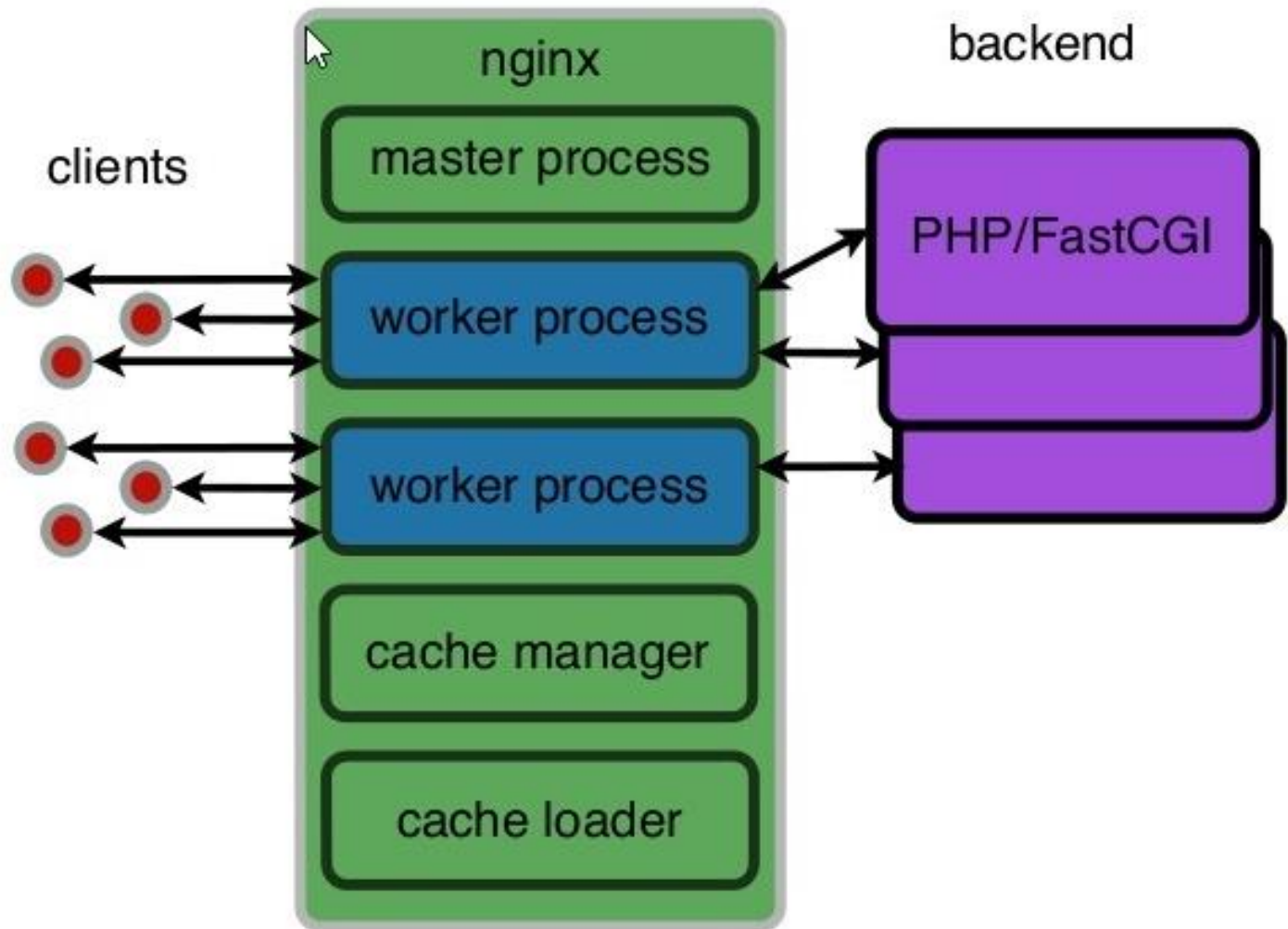
Web Server Architecture



Traditional (Blocking) Thread Model

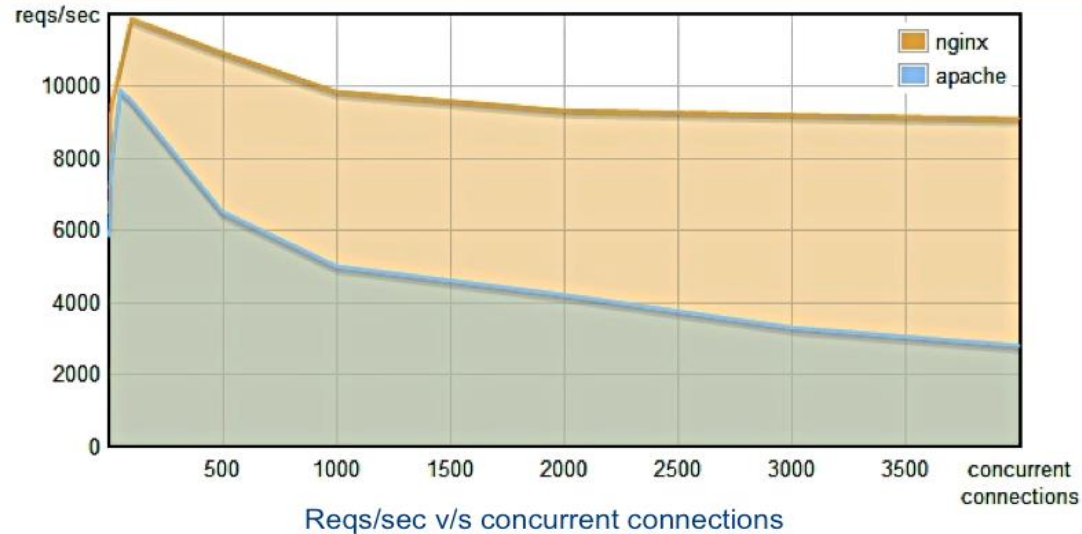


Nginx



Apache vs Nginx

Apache V/s Nginx: performance



At ~4000 concurrent connections,
- Nginx can serve ~9000 reqs/sec
- Apache can serve ~3000 reqs/sec

Ref: <http://blog.webfaction.com/a-little-holiday-present>

Picture source : cloudfoundry, vmware

Cost of IO

The diagram illustrates the cost of IO for various components, categorized into Non-Blocking IO and Blocking IO. The components and their respective costs are listed in a table-like format, with blue curly braces grouping them into two categories.

▪ L1-cache	3 cycles	Non-Blocking IO
▪ L2-cache	14 cycles	
▪ RAM	250 cycles	
▪ Disk	41 000 000 cycles	Blocking IO
▪ Network	240 000 000 cycles	

Handling IO

- Threads wait for IO to complete at application level.

Example

```
results = db.getData('select * from users');
```

Evented asynchronous platform

Javascript in browser

- Single Threaded.
- Asynchronous.
- Functional language. Native callback function support.

Asynchronous IO

Example

```
db.getData('select * from users', function(err,  
results) {  
    console.log(results);  
});
```

Non-Blocking I/O

- Traditional I/O

```
var result = db.query("select x from table_x");  
doSomethingWith(result); //wait for result!  
doSomethingWithoutResult(); //execution is blocked!
```

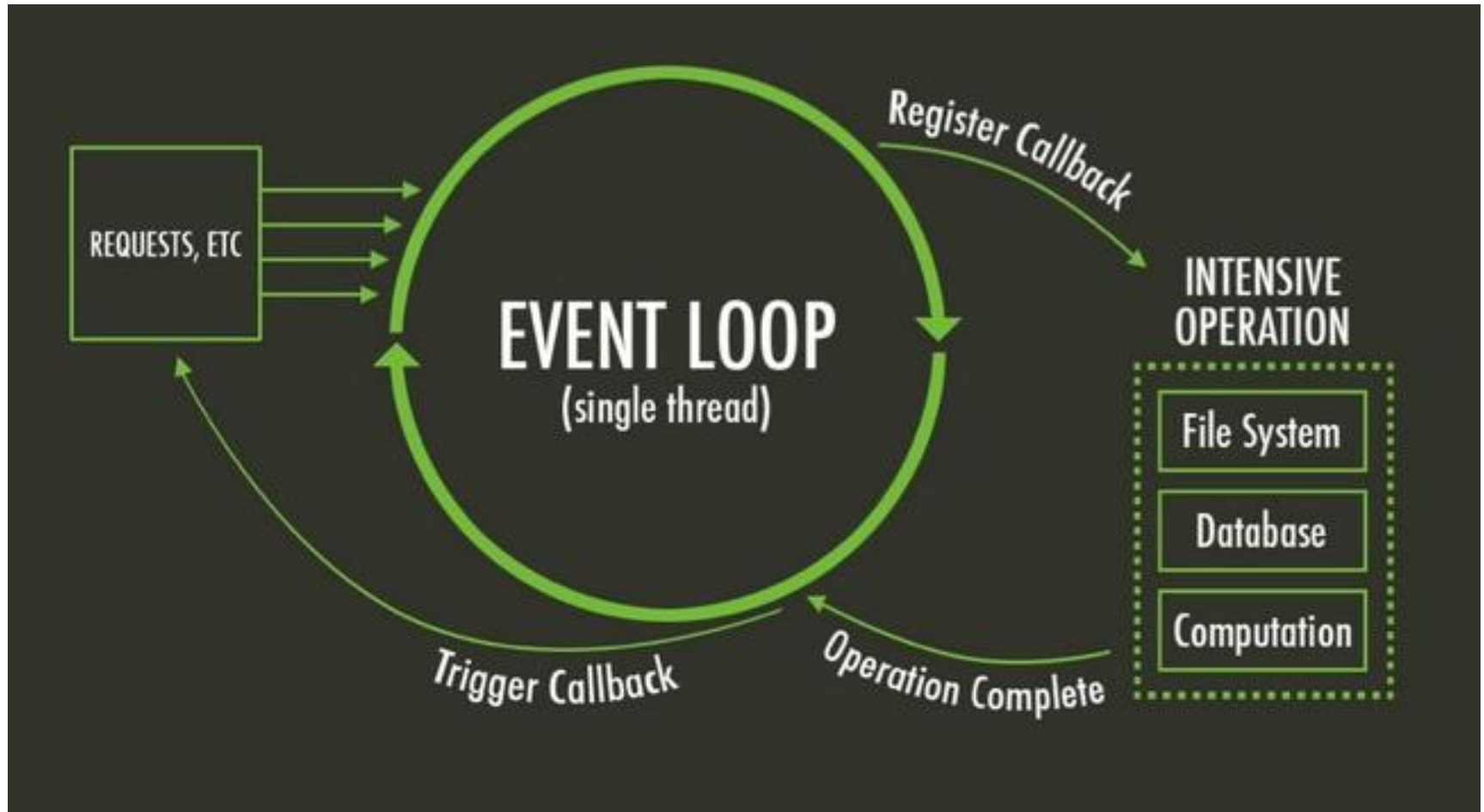
- Non-traditional, Non-blocking I/O

```
db.query("select x from table_x",function (result){  
    doSomethingWith(result); //wait for result!  
});  
doSomethingWithoutResult(); //executes without any delay!
```

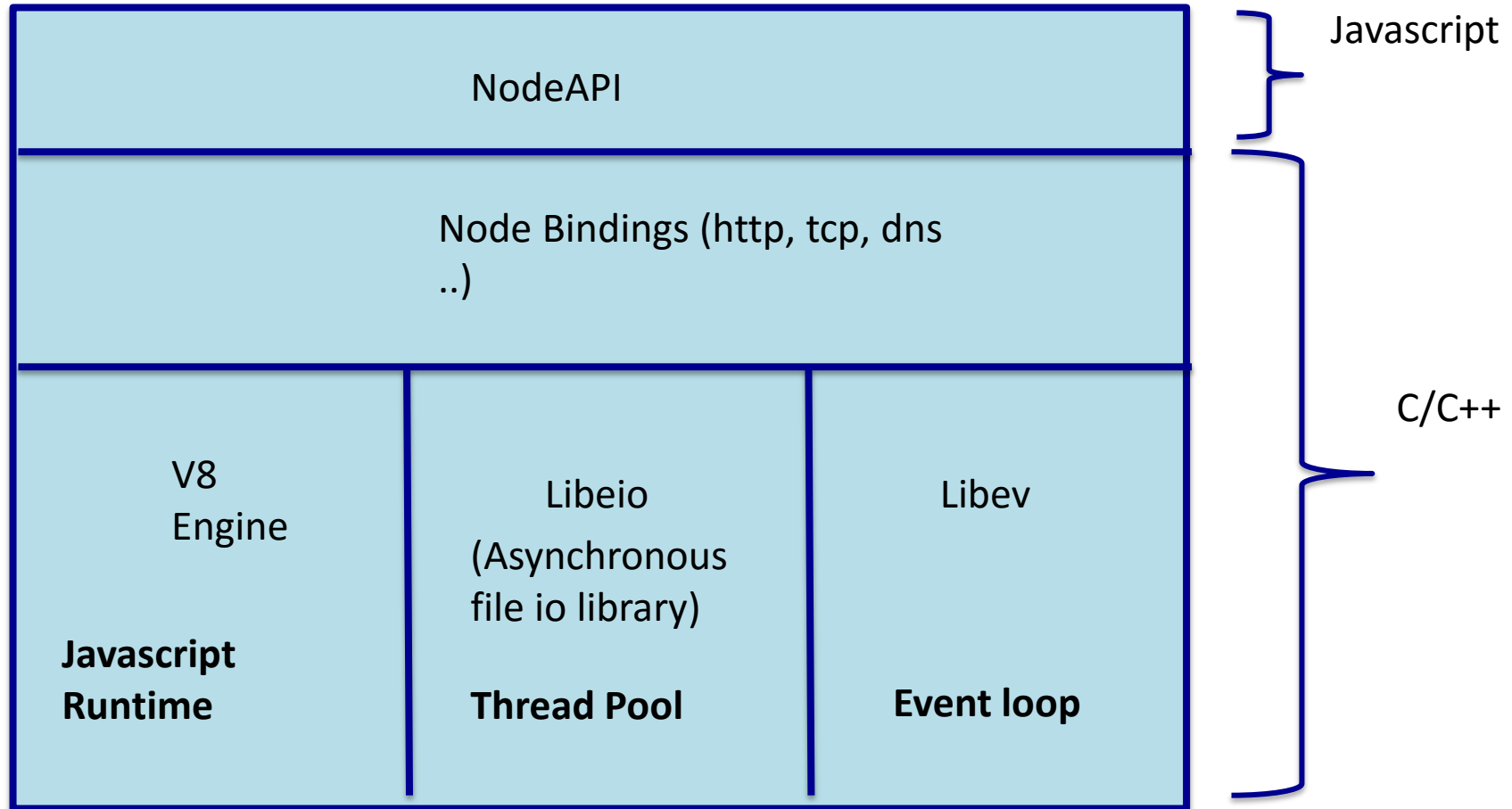
Introduction

- "Node's goal is to provide an easy way to build **scalable network programs**" : (from nodejs.org)
- Node.js is a high-performance **network applications framework**, well optimized for high concurrent environments.
- Everything inside Node.js runs in a **single-thread**.
- Node.js uses an **event-driven, non-blocking I/O** model, which makes it lightweight.
- It makes use of **event-loops** via JavaScript's callback functionality to implement the **non**

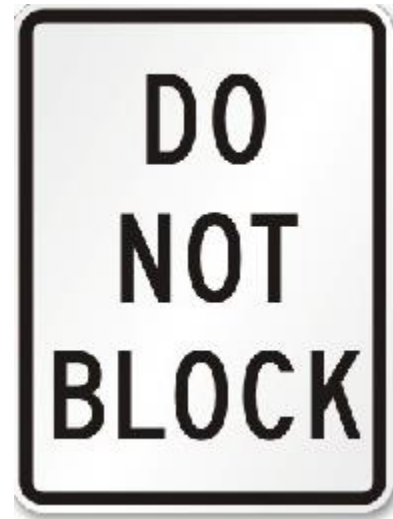
Event Loop Example



Node.js under the hood



Non-blocking I/O



- Servers do nothing but I/O
 - Scripts waiting on I/O requests degrades performance
- To avoid blocking, Node makes use of the event driven nature of JS by attaching callbacks to I/O requests
- Scripts waiting on I/O waste no space because they get popped off the stack when their non-I/O related code finishes executing

Cool facts

- Javascript is the most popular language in Github. (132K repositories)
- node.js third most starred repository in Github. (29K)
- One of the most quickly adopted platform (1-2 years)
- Companies who gained from using nodejs (linkedin, paypal,groupon, walmart labs, uber, airbnb and many gaming companies)

When to use Node.js

- Node.js is good for creating streaming based real-time services, web chat applications, static file servers etc.
- If you need high level concurrency and not worried about CPU-cycles.
- If you are great at writing JavaScript code because then you can use the same language at both the places: server-side and client-side.

Resources to Get Started

- Watch this video at Youtube:
http://www.youtube.com/watch?v=jo_B4LTHi3I
- Read the free O'reilly Book '**Up and Running with Node.js**'
- Visit www.nodejs.org for Info/News about Node.js
- Watch Node.js tutorials @
<http://nodetuts.com/>
- For anything else Google!

References

- <http://nodejs.org/>
- <http://nodejs.org/cinco de node.pdf>
- <http://ajaxian.com/archives/google-chrome-chromium-and-v8>
- <http://blog.chromium.org/2010/12/new-crankshaft-for-v8.html>
- <http://news.softpedia.com/news/IE9-RC-vs-Chrome-10-9-vs-Opera-11-vs-Firefox-11-Performance-Comparison-183973.shtml>

Node.js Codes

What is a MEAN stack?

- [MongoDB](#) as the database
 - [Express](#) as the web framework
 - [AngularJS](#) as the frontend framework, and
 - [Node.js](#) as the server platform
-
- Single language is used in the whole application
 - Support for the [MVC pattern](#)
 - [JSON](#) is used for transferring data
 - Node.js's huge module library

Node Importing Modules

- Java/ Python use “import” to load other libraries
- PHP/ Ruby use “require” to load libraries
- Node.js similarly uses “require” to load other dependencies

`//Loading external modules`

- `var http = require('http');`

`//Loading relative files from the project`

- `var myFile = require('./myFile');`
- External modules can be installed locally using npm command:

- `npm install express` `//to install express module`

Modules

- A Module encapsulates related code in a single unit

- `Authenticate.js` – Module

```
var sign_in = function(req,res) { //functionality goes here }  
var sign_up = function(req,res) { //functionality goes here }
```

- **Exporting** `Authenticate.js` module

```
exports.sign_in = function(req,res) { //functionality goes here }  
exports.sign_up = function(req,res) { //functionality goes here }
```

- Importing `Authenticate.js` module

```
var authenticate = require("./authenticate.js");  
authenticate.sign_in(req,res);
```

Callback functions

- Node.js uses JavaScript, which has callback functions
- Normal functions wait for the function block to complete
- Callback function is a function called in another function as a parameter and is called inside the function

```
var customCallback = function(data) {  
    console.log('Data is : '+data);  
};  
  
var checkTheCallback = function(callback) {  
    callback('Hello World!');  
};
```

```
checkTheCallback(customCallback)
```

- Callback function enables node.js in asynchronous, non-blocking implementation

• Blocking Code

```
var contents = fs.readFileSync('/etc/hosts');  
console.log(contents);  
console.log('Doing something else');
```

Stop process until complete



• Non-Blocking Code

```
fs.readFile('/etc/hosts', function(err, contents) {  
  console.log(contents);  
});  
console.log('Doing something else');
```



Simple HTTP Server

```
var http = require("http");           //load external
library - http

http.createServer(function (req,res) { //callback
function
    res.writeHead(200,
        {"Content-Type": "text/plain"});
    res.sendBody("Hello\r\n");
    res.sendBody("World\r\n");
    res.finish();
}).listen(8000);                       //server connected and listening
to the port 8000
```

Simple HTTP Server

- node http_server.js

with the filename

//Running node command

- curl -i <http://localhost:8000>

to check output

//Running curl command

//Expected Output

HTTP/1.1 200 OK

Content-Type: text/plain

Connection: keep-alive

Transfer-Encoding: chunked

Hello

World

OR

- <http://localhost:8000>

browser

//Run the link in your

Req and Res in Node.js

- `req = { _startTime : Date,
 app : function(req,res){},
 body : {},
 client : Socket,
 complete : Boolean,
 connection : Socket,
 cookies : {},
 files : {},
 headers : {},
 httpVersion : String
 method : String, // e.g. GET POST PUT
 DELETE next : function next(err){}, originalUrl
 : String, /* e.g. /erer?param1=23¶m2=45 */
 params : [],
 query : {},
 readable : Boolean,
 route : Route,
 signedCookies : {},
 socket : Socket,
 url : String /*e.g. /erer?param1=23¶m2=45
 */ }`
- `res = { app : function(req, res) {},
 chunkedEncoding: Boolean,
 connection : Socket,
 finished : Boolean,
 output : [],
 outputEncodings: [],
 req : IncomingMessage,
 sendDate : Boolean,
 shouldkeepAlive : Boolean,
 socket : Socket,
 useChunkedEncdoingByDefault : Boolean,
 viewCallbacks : [],
 writable : Boolean }`

Express Framework, Connect

- **Express.js** is a Node.js web application server framework, designed for building single page, multi-page, and hybrid web applications
- Express is the backend part of the MEAN stack, together with MongoDB database and AngularJS frontend framework
- Sinatra-inspired MVC framework for Node.JS
- Built on Connect Middleware
- **Connect** is an extensible HTTP server framework for node using "plugins" known as middleware

What Express Does?

- Parses arguments and headers
- Routing
- Views
 - Partials
 - Layouts
- Configurations
- Sessions

Express 4.0

```
var express = require('express');           // call express
var app = express();                         // define our app
using express
var bodyParser = require('body-parser');

// configure app to use bodyParser()
// this will let us get the data from a POST
app.set('views', __dirname + '/views');      //setting the path for
the views
app.set('view engine', 'ejs');               //setting the view
engine to ejs
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
var port = process.env.PORT || 8080;         // set our port

var router = express.Router();
// respond with "Hello World!" on the homepage
router.get('/', function(req, res)
{
    res.send('Hello World!');
});
app.use('/api', router);
// START THE SERVER
app.listen(port);
console.log('Express server listening on port ' + port);
```

Express Configuration file (app.js)

```
var express = require('express');           // loading modules
var app = express();                         // initializing express
var bodyParser = require('body-parser');     // loading body-parser
// configure app to use bodyParser()
// this will let us get the data from a POST
app.set('views', __dirname + '/views'); //setting the path for the views
app.set('view engine', 'ejs');             //setting the view engine to ejs
app.use(sessions({secret: 'adfasdf34efsdffs34sefsdf'})); //setting the
session key
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(express.static(__dirname + '/public')); //setting path for
static(images, stylesheets)
var port = process.env.PORT || 8080;        // set our port
app.configure('development', function() {   //development-environment
configurations
    app.use(express.errorHandler({ dumpExceptions: true, showStack:
true }));
});
//Start Server
app.listen(port);
console.log('Express server listening on port ' + port);
```

Routing

```
//Catch-all
```

```
app.all('/app(/*)?', requiresLogin);           //works for all HTTP  
verbs
```

```
// Routes
```

```
app.get('/', routes.index);    //GET Request, for the homepage
```

```
app.get('/about', routes.about); //GET Request, for other routes
```

```
app.get('/contact', routes.contact);
```

```
app.get('/app/list', routes.listapps);
```

```
app.get('/app/new', routes.newapp);
```

```
app.post('/app/new', routes.saveapp);           //POST Request
```

```
app.get('/app/:app', routes.getapp);
```

```
app.get('/app/:app/edit', routes.editapp);
```

Syntax pattern: `App.[verb](path, function(req,res),
[function(req,res)]);`

How about /user/12?

Request Object

- **req.param**
 - Return the value of param name when present
 - **req.param** is an abstraction layer for picking up information about a request – it automatically searches:
 - Query strings
 - Posted form values
 - Route values
- **req.session**
 - To store or access session data
- **req.params**
 - object containing properties mapped to the named route “parameters”
 - Eg: `/user/:name`, then the “**name**” property is available as `req.params.name`
- **req.headers**
 - Returns the specified HTTP request header field (case-insensitive match)

Response Object

- **res.render**

- Renders a view and sends the rendered HTML string to the client

```
// send the rendered view to the client
```

```
res.render('index');
```

```
//send the rendered view to the client with the parameters
```

```
res.render('index', {name: "SJSU"});
```

- **res.end**

- Ends the response process

- **res.redirect**

- Redirects to the URL derived from the specified path
- Redirects can be a fully-qualified URL for redirecting to a different site

```
res.redirect('http://google.com');
```

- Redirects can be relative to the root of the host name

```
res.redirect('/admin');
```

Views

- Support for multiple view engines
 - Jade
 - **Ejs**
 - Jshtml
 - Hogan-js
- Layout supports
- Partial
- Dynamic Helpers

Session Management

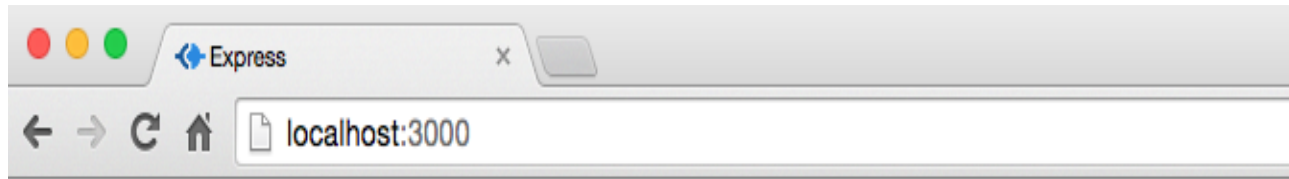
- **Session State Providers**
 - Cookie + Back-end Session Store
- **Session Cookies**
 - cookie-sessions NPM package

```
//store the username and email address after  
successful login  
  
req.session.username = username;  
req.session.email_address = email_address;
```

Example 1

- Simple Login Application, to check the username and password.
- The user should be directed to different pages on validating.
- Incorrect username, password entered should be directed to different page and valid login should direct to different page

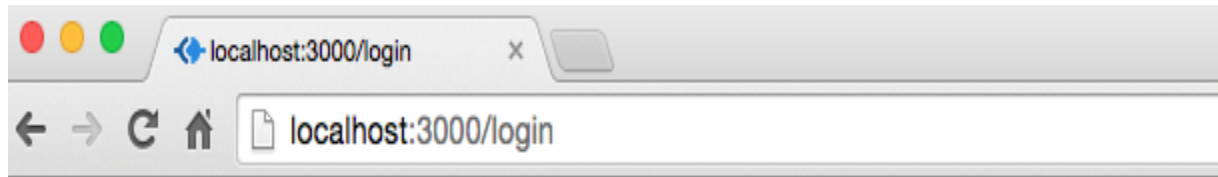
Example 1 – Login Page



Username:

Password:

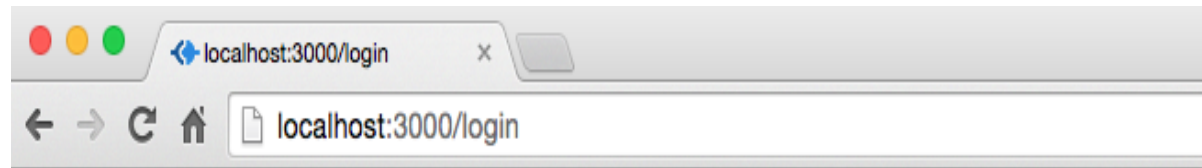
Example 1 – Success Page



Welcome to the Portal, test1

[Back](#)

Example 1 – Error Page



Incorrect username, password

[Back](#)

Example 1 – app.js

```
1
2 /**
3  * Module dependencies.
4  */
5
6 var express = require('express')
7   , routes = require('./routes')
8   , login = require('./routes/login')
9   , http = require('http')
10  , path = require('path');
11
12 var app = express();
13
14 // all environments
15 app.set('port', process.env.PORT || 3000);
16 app.set('views', __dirname + '/views');
17 app.set('view engine', 'ejs');
18 app.use(express.favicon());
19 app.use(express.logger('dev'));
20 app.use(express.bodyParser());
21 app.use(express.methodOverride());
22 app.use(app.router);
23 app.use(express.static(path.join(__dirname, 'public')));
24
25 // development only
26 if ('development' == app.get('env')) {
27   app.use(express.errorHandler());
28 }
29
30 app.get('/', routes.index);
31 app.post('/login', login.login);
32
33 http.createServer(app).listen(app.get('port'), function(){
34   console.log('Express server listening on port ' + app.get('port'));
35 });
36
```

Example 1 – login.js

```
1 /**
2  * New node file
3  */
4 exports.login = function(req,res)
5 {
6     var username, password;
7     username = req.param("username");
8     password = req.param("password");
9
10    console.log(username+" "+password);
11    if(username === "test1" && password === "test1")
12    {
13        res.render("success", {username:username});
14    }
15    else
16    {
17        res.render("error");
18    }
19 };
```

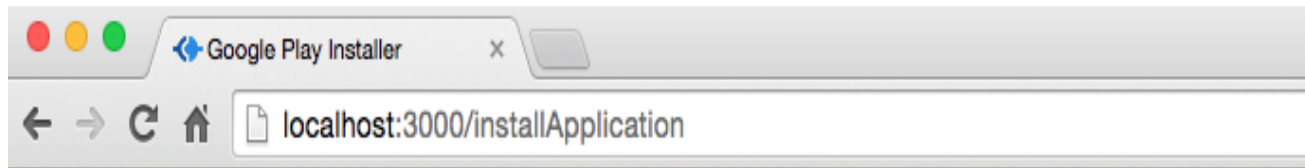
Example 1 – index.ejs

```
1 <!DOCTYPE html>
SampleApplication/app.js
3 <head>
4 <title><%= title %></title>
5 <link rel='stylesheet' href='/stylesheets/style.css' />
6 </head>
7 <body>
8 <form method="post" action="login">
9     Username:<br> <input type="text" name="username"> <br>
10    Password:<br> <input type="password" name="password">
11    <input type="submit" value="Login"/>
12 </form>
13 </body>
14 </html>
```

Example 2

- Google Play installer application
- User selects an application from the list available
- Application selects whether the application exists
- If it exist is checks whether it is already installed
- Else it gives the message for successful install

Example 2 – Application Page



Google Play Store Application Installer

Application Name :

- ✓ Install Application
- Facebook
- Gmail
- Twitter
- Whatsapp
- YouTube
- FourSquare
- Pinterest
- Google Maps
- Wave
- Gallery

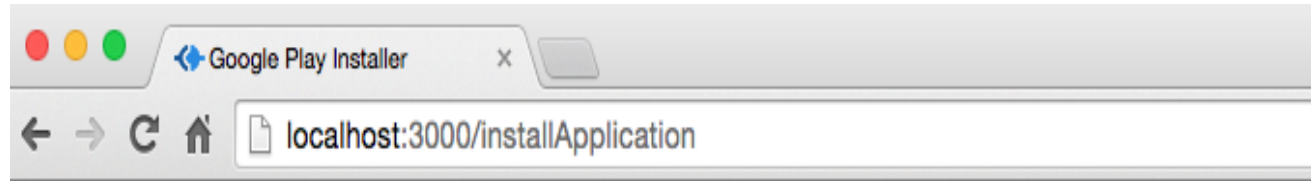
Install

ady installed

S

l Whatsapp

Example 2 – Application Installed



Google Play Store Application Installer

Application Name :

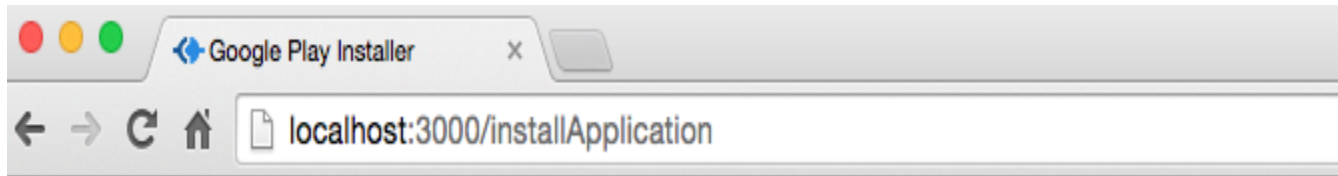
Install Application

Facebook installed successfully

Installed Applications

|| Gmail || Whatsapp || FourSquare || Facebook

Example 2 – Application Already Installed



Google Play Store Application Installer

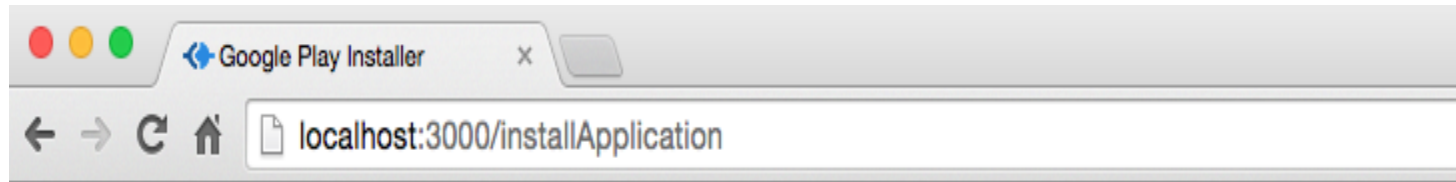
Application Name :

Facebook is already installed

Installed Applications

|| Facebook || Gmail || Whatsapp

Example 2 – Application not found



Google Play Store Application Installer

Application Name :

Install Application



Install

Gallery Application not found

Installed Applications

|| Twitter || Gmail || FourSquare

Example 2 – index.ejs

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Google Play Installer</title>
5 </head>
6 <body>
7
8 <h1>Google Play Store Application Installer</h1>
9 <br /> Application Name :
10 <form action="installApplication" method="post">
11 <select id="applicationInstall" name="applicationInstall">
12 <option value="InstallApplication">Install Application</option>
13 <option value="Facebook">Facebook</option>
14 <option value="Gmail">Gmail</option>
15 <option value="Twitter">Twitter</option>
16 <option value="Whatsapp">Whatsapp</option>
17 <option value="YouTube">YouTube</option>
18 <option value="FourSquare">FourSquare</option>
19 <option value="Pinterest">Pinterest</option>
20 <option value="Google Maps">Google Maps</option>
21 <option value="Wave">Wave</option>
22 <option value="Gallery">Gallery</option>
23 </select> <input type="submit" value="Install" />
24 </form>
25
26 <p id="installStatus"></p>
27
28 <h3><b>%= message %</b></h3>
29 Installed Applications
30 <br /> -----
31 <p>%= applicationsInstalled%</p>
32 </body>
33 </html>
34
```

Example 2 – app.js

```
1
2 /**
3  * Module dependencies.
4  */
5
6 var express = require('express')
7   , routes = require('./routes')
8   , user = require('./routes/user')
9   , http = require('http')
10  , path = require('path');
11
12 var app = express();
13
14 // all environments
15 app.set('port', process.env.PORT || 3000);
16 app.set('views', __dirname + '/views');
17 app.set('view engine', 'ejs');
18 app.use(express.favicon());
19 app.use(express.logger('dev'));
20 app.use(express.bodyParser());
21 app.use(express.methodOverride());
22 app.use(app.router);
23 app.use(express.static(path.join(__dirname, 'public')));
24
25 // development only
26 if ('development' == app.get('env')) {
27   app.use(express.errorHandler());
28 }
29
30 app.get('/', routes.index);
31 app.post('/installApplication', validator.installApplication);
32
33
34 http.createServer(app).listen(app.get('port'), function(){
35   console.log('Express server listening on port ' + app.get('port'));
36 });
37
```

Exercise (show demo next week)

- Build a sign up page with fields username, password, firstname, lastname, date of birth, gender
- Build a login page to login
- Build 2 pages for after successful login and incorrect password/username
- The after successful login page should show all the information of the user(firstname, lastname, date of birth and gender)
- The error page should show the message and take him back to the login page

Future References

- [ExpressJS.com](https://expressjs.com) – Official Express JS Homepage
- docs.npmjs.com – Documentation for npm
- nodejs.org/api – Node.js API Documentation