Express

Agenda

- QA
- Modern WEB overview
- Web architecture, Http
- HTTP module
- Express.js
- QA

QA

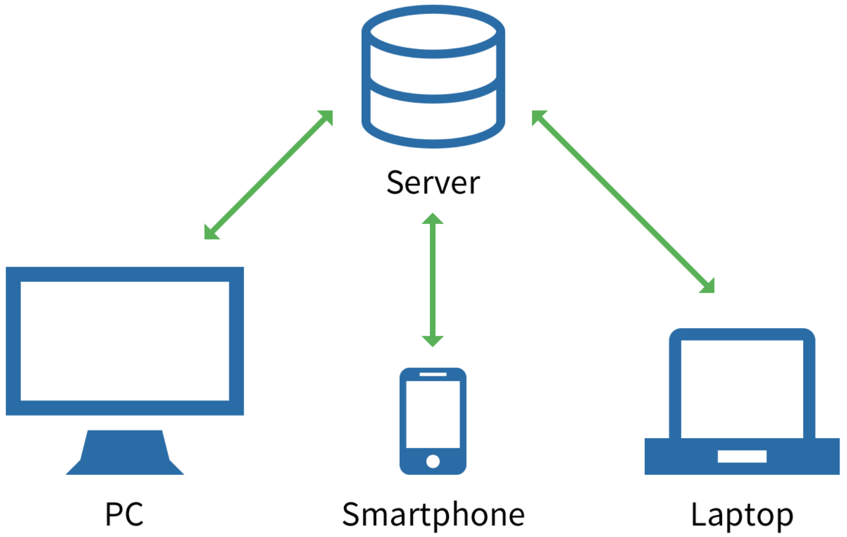
MODERN WEB OVERVIEW

Client-server architecture

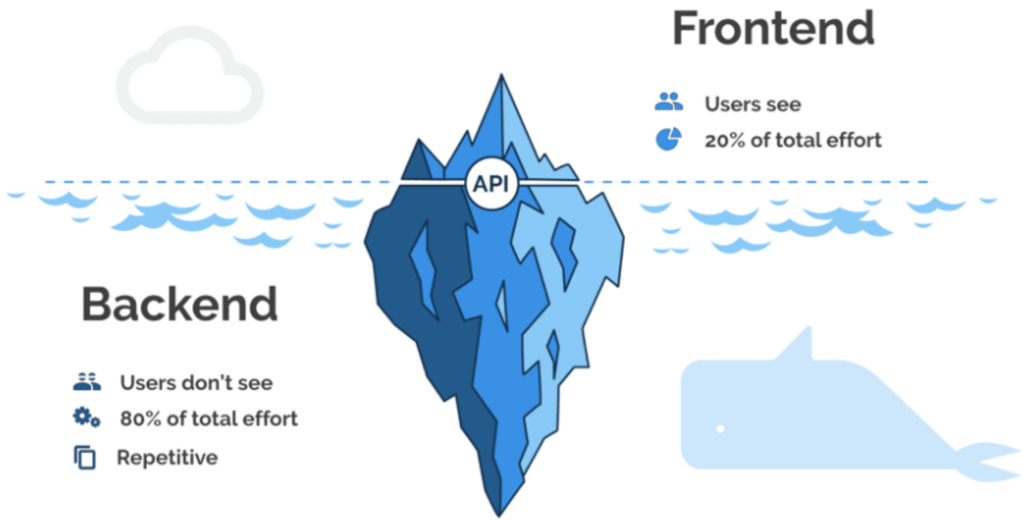
Client/server architecture is a producer/consumer computing architecture where the server acts as the producer and the client as a consumer.

The server houses and provides high-end, computing-intensive services to the client on demand.

These services can include application access, storage, file sharing, printer access and/or direct access to the server's raw computing power.



Client-server architecture

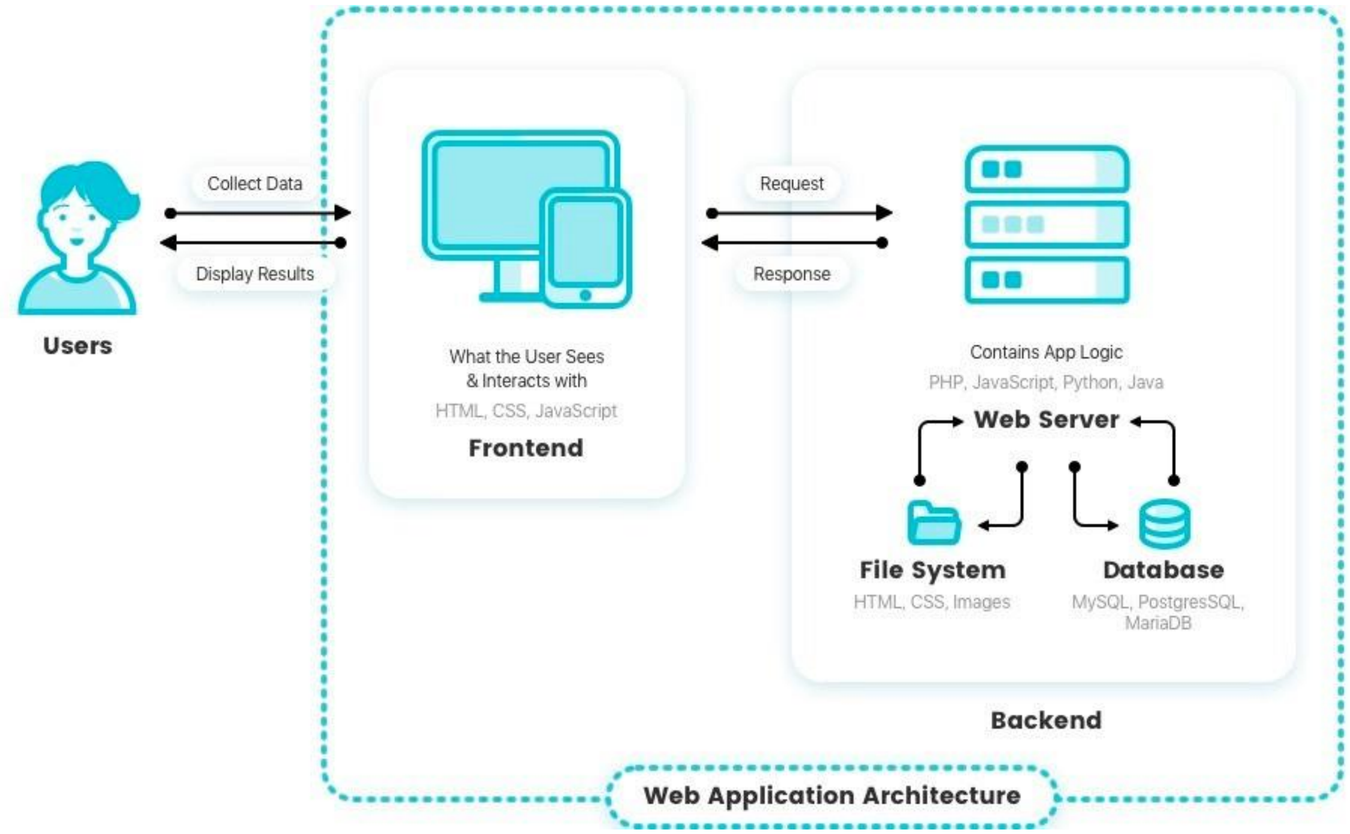


Client/server architecture works when the client computer sends a resource or process request to the server over the network connection, which is then processed and delivered to the client.

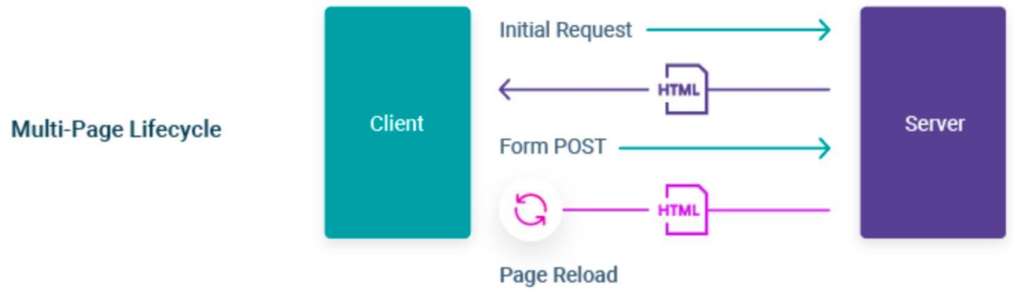
A server computer can manage several clients simultaneously, whereas one client can be connected to several servers at a time, each providing a different set of services.

In its simplest form, the internet is also based on client/server architecture where web servers serve many simultaneous users with website data.

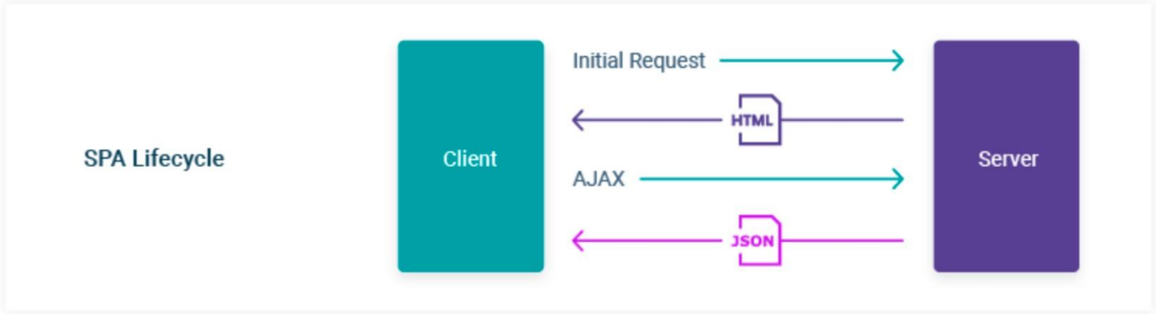
Modern web applications



MPA vs SPA

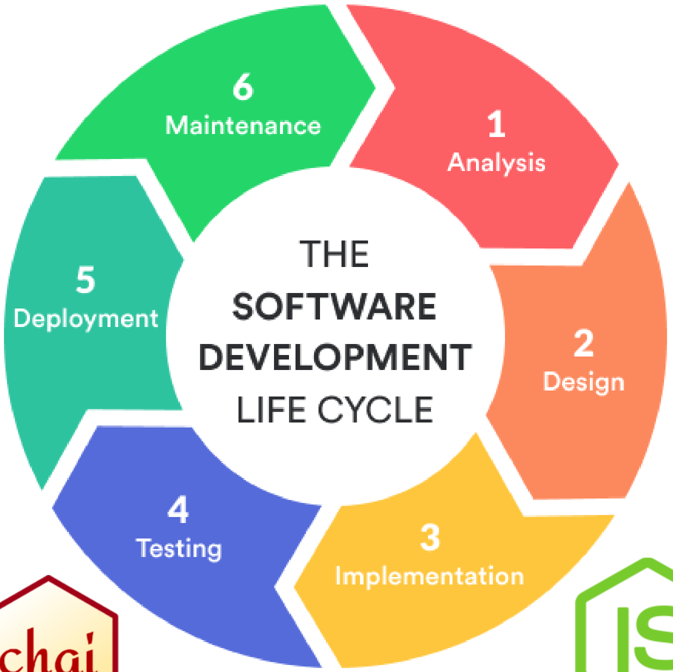


Multi-Page Lifecycle
Client
Initial Request
HTML
Form POST
HTML
Page Reload
Server



SPA Lifecycle
Client
Initial Request
HTML
AJAX
JSON
Server

Modern web applications



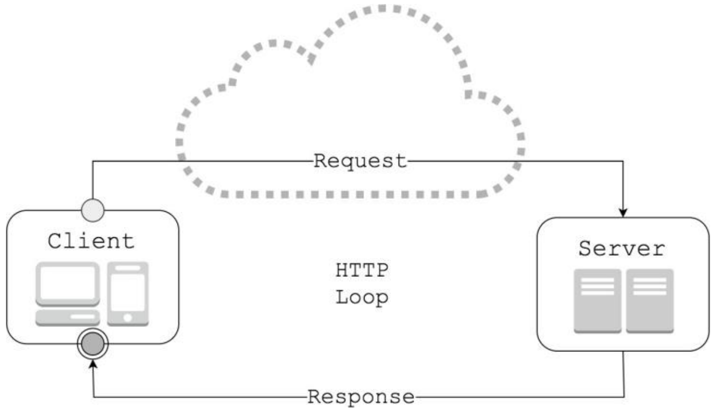
HTTP

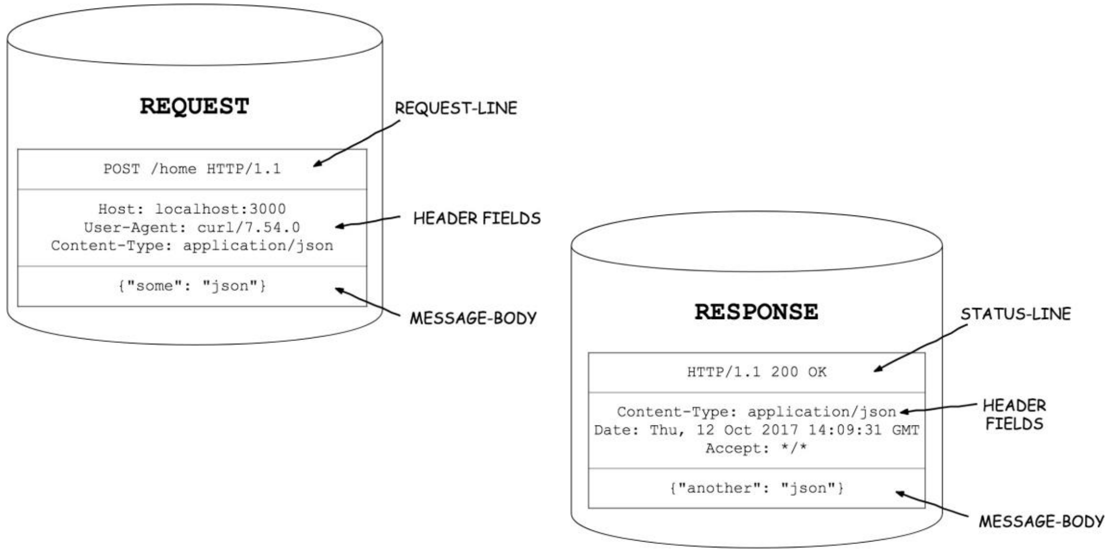
HTTP

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems.

This is the foundation for data communication for the World Wide Web (i.e. internet) since 1990.

HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers.





```
POST /home HTTP/1.1

Host: localhost:3000
User-Agent: curl/7.54.0
Content-Type: application/json

{"some": "json"}
```

```
HTTP/1.1 200 OK

Content-Type: application/json
Date: Thu, 12 Oct 2017 14:09:31 GMT
Accept: */*

{"another": "json"}
```

URI and URL

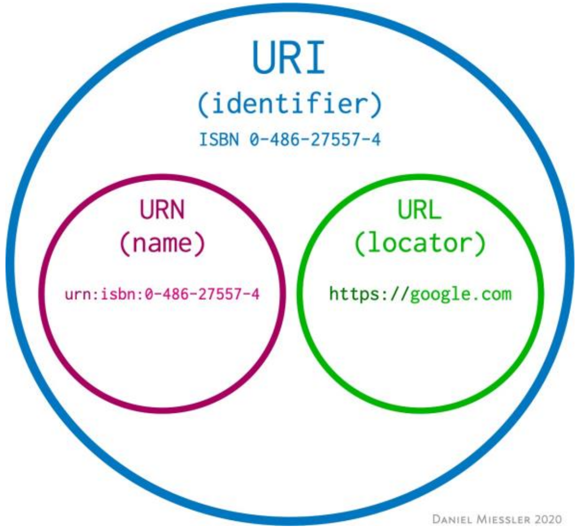
Uniform Resource Identifier

Uniform Resource Locator (Browser)

A Uniform Resource Identifier (URI) provides a simple and extensible means for identifying a resource (straight from RFC 3986). It's just an identifier; don't overthink it.

A URI is an identifier.

A URL is an identifier that tells you how to get to it.



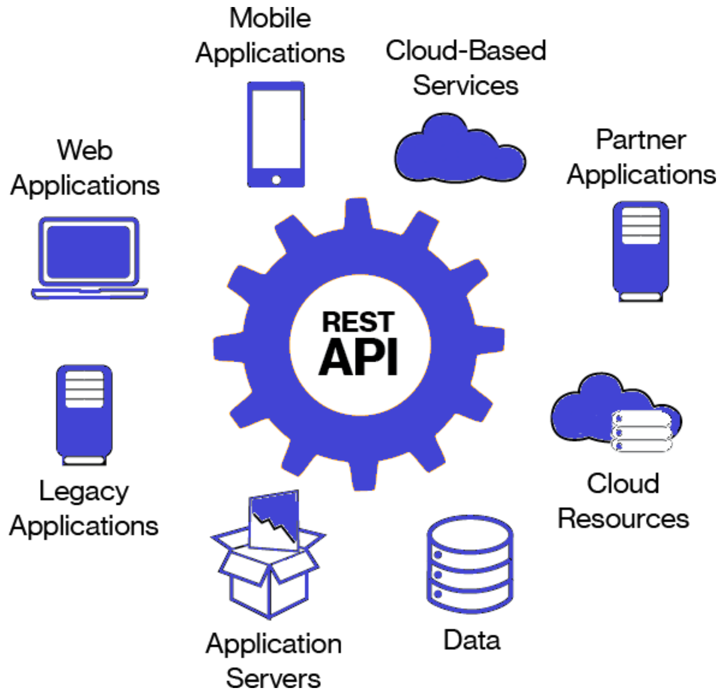
Mime types

```
.css - text/css
.csv - text/csv
.gif - image/gif
.html - text/html
.js - text/javascript
.json - application/json
.txt - text/plain
.xml - application/xml
Url encoded - application/x-www-form-urlencoded
```

HTTP, RESTful API

REST is acronym for **RE**presentational **S**tate **T**ransfer. It is architectural style for **distributed hypermedia systems.**

The key abstraction of information in REST is a **resource**. Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person), and so on. REST uses a **resource identifier** to identify the particular resource involved in an interaction between components.



HTTP MODULE

HTTP module

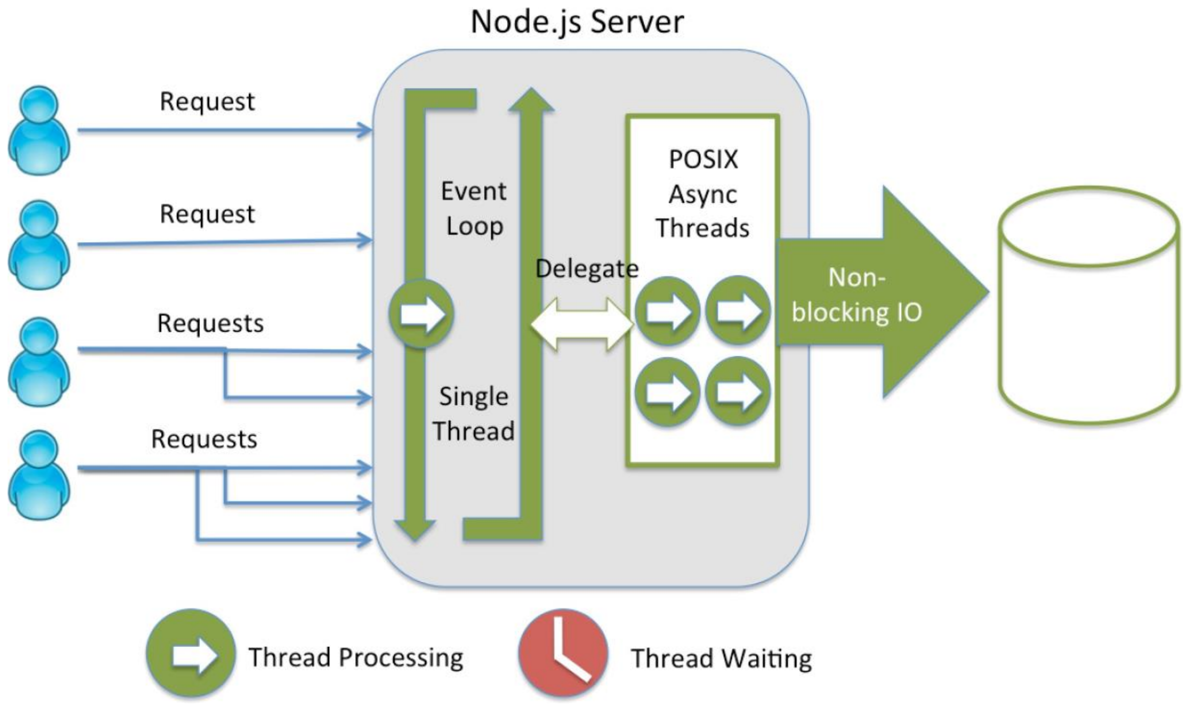
Built-in HTTP Module

```
var http = require(‘http');
```

Now your application has access to the HTTP module,
and is able to create a server:

```
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/
html'});
  res.end('Hello World!');
}).listen(8080);
```

Web server architecture



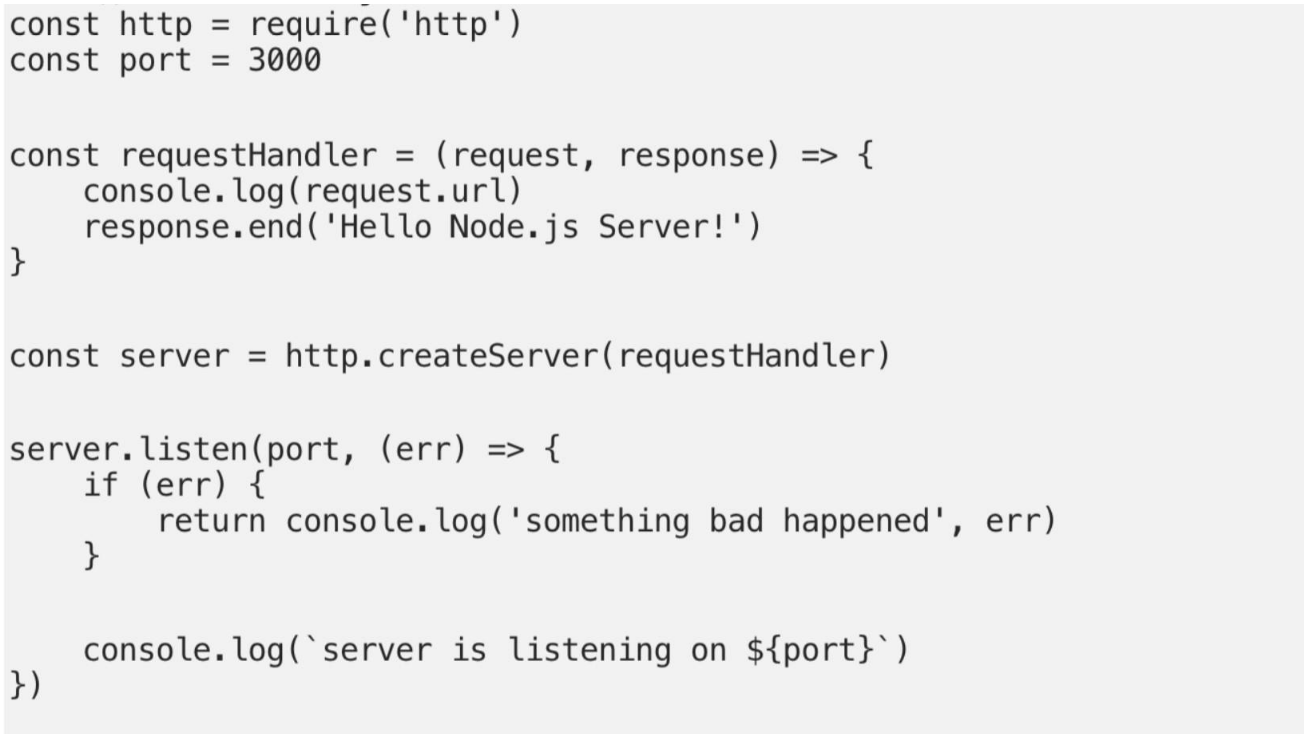
EXPRESS.JS

Web frameworks

node JS ™

Express koa sails

HTTP module in Node.JS

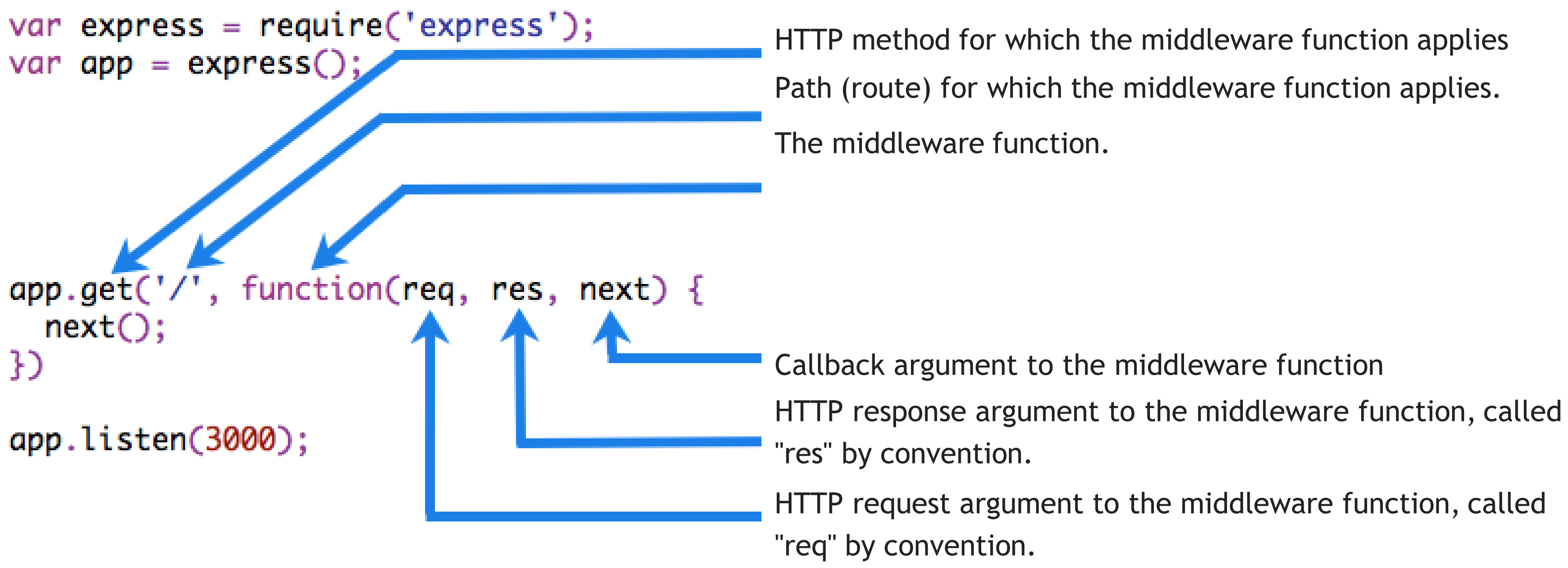


```
const http = require('http')
const port = 3000


const requestHandler = (request, response) => {
    console.log(request.url)
    response.end('Hello Node.js Server!')
}


const server = http.createServer(requestHandler)


server.listen(port, (err) => {
    if (err) {
        return console.log('something bad happened', err)
    }


    console.log(`server is listening on ${port}`)
})
```

Express.js

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

app.get('/', function (req, res) {
    res.json({ ok: true });
});

app.listen(3000);
```

Routing

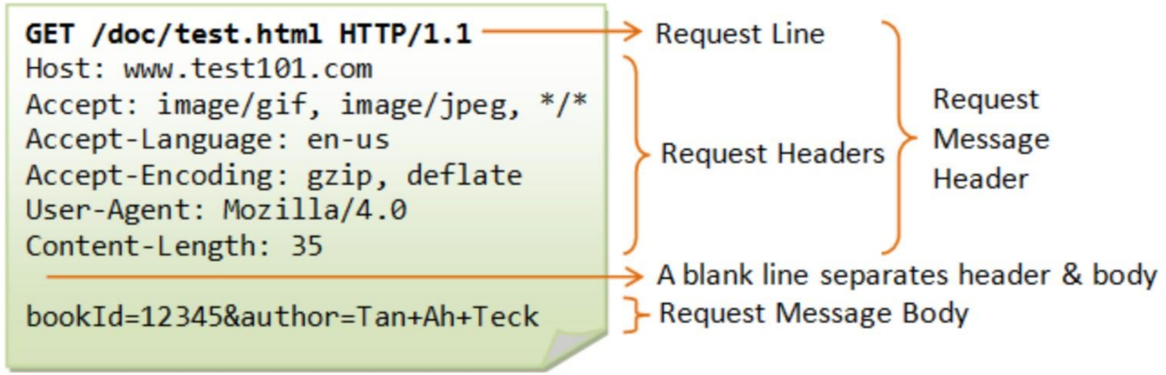


```
var express = require('express');
var app = express();

app.get('/', function(req, res, next) {
  next();
})

app.listen(3000);
```

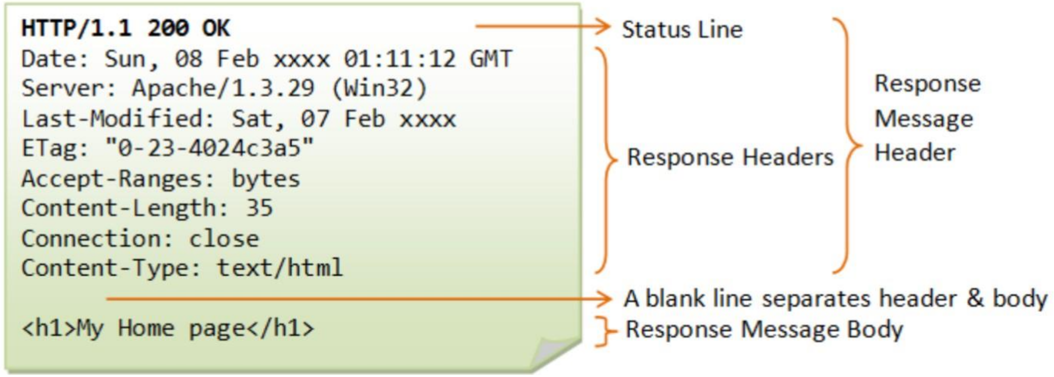
HTTP method for which the middleware function applies
Path (route) for which the middleware function applies.
The middleware function.
Callback argument to the middleware function
HTTP response argument to the middleware function, called "res" by convention.
HTTP request argument to the middleware function, called "req" by convention.

Express.js 4.x API reference

- Express
- Application
- Request
- Response



```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```



```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

Response

- Represents the HTTP response that an Express app sends for HTTP request
- Sending response:

res.end()
res.sendStatus()
res.send()
res.sendFile()

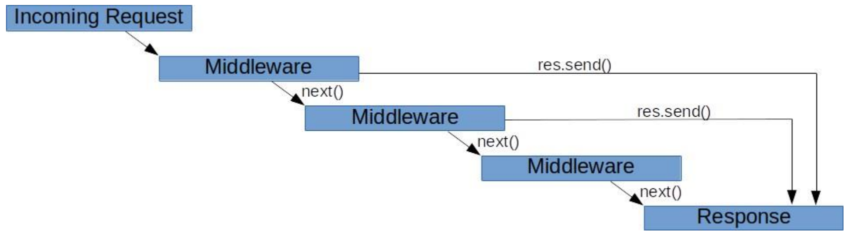
res.json()
res.redirect()
res.render()

What is Express

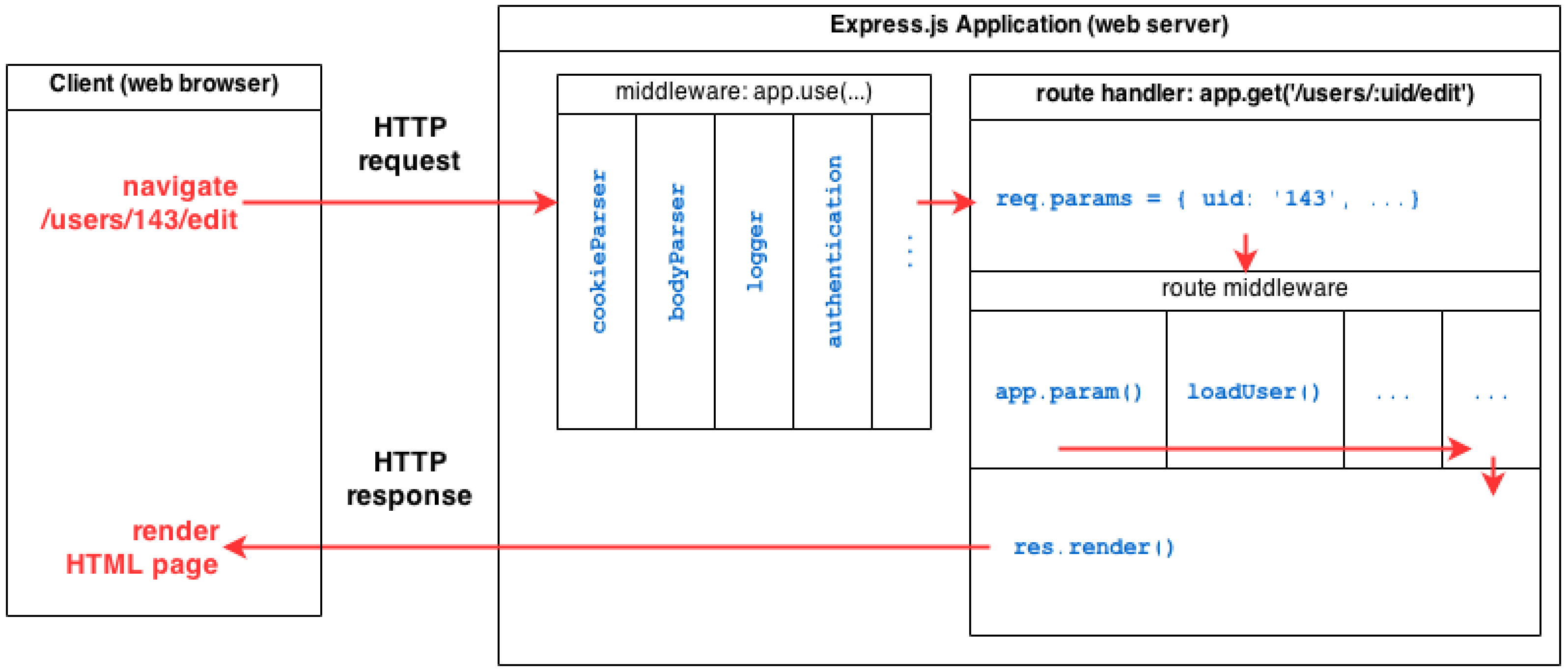
Express = Routing + Middlewares

Middleware

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



REQUEST -> RESPONSE CYCLE



Example

```
app.put('/employees/:id', async (req, res) => {
    const employee = await getEmployeeById(req.params.id);

    if (!employee) {
        return res.status(400).json({message: ‘No employee found'});
    }

    await employee.update(req.body);
    res.json({ status: 'ok' });
});
```

Request: body

- Contains submitted data as key-value pairs and undefined by default
- Use **express.json()** middleware to populate body from json

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

app.use(express.json());
```

Static, built-in middleware

To serve static files such as images, CSS files, and JavaScript files, use the `express.static` built-in middleware function in Express.

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

External middleware

Morgan - HTTP request logger middleware for node.js

```
var express = require('express')
var morgan = require('morgan')

var app = express()

app.use(morgan('combined'))

app.get('/', function (req, res) {
  res.send('hello, world!')
})
```

Router

```
const express = require('express');
const app = express();
const router = express.Router();

router.use((req, res, next) => {
    // some middleware
    next();
});

router.get('/:id', (req, res) => {
    res.json({ id: req.params.id })
});

app.use('/users', router);
```

nodemon

QA