EXPRESS.JS

HTTP REQUEST

just a message with a certain format

```
POST /docs/1/related 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 (compatible; MSIE 6.0; Windows NT 5.1)

bookId=12345&author=Nimit
```

(from http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html)

HTTP requests and responses are just strings of text in a certain format. But because these strings adhere to this format, we know where to look to find things like the verb, the uri, the headers and the body. This means we can programmatically interpret HTTP requests using some simple string manipulation.

Possibly show: `curl -D - http://www.google.com`

“Oh boy! Parsing this string of text will be lots of fun!”

– NO DEVELOPER EVER

…But this would suck

BUILDING A SERVER WITH NODE

- Node.js provides several low-level tools for building servers
- Abstracts HTTP requests and and responses into nice objects that have useful methods

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

const server = http.createServer()

server.listen(3000, 'localhost')
```


The built in http module is already at a high level of abstraction in that it gives us an easy way to “listen” for http traffic coming to our OS on a specific port

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

const server = http.createServer()

server.on('request', (req, res) => {
  res.writeHead(200)
  res.write('<h1>Welcome to my website</h1>')
  res.end()
})

server.listen(3000, 'localhost')
```


We can tell our server that when it gets ANY request, it can send some kind of response using the res object.

But we don’t want to send the same response for every request…

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

const server = http.createServer()

server.on('request', (req, res) => {
  if (req.method === 'GET') {
    if (req.url === '/') {
      res.writeHead(200)
      res.write(`<h1>Welcome to the main page!</h1>`)
      res.end()
    } else if (req.url === '/puppies') {
      res.writeHead(200)
      res.write(`<h1>Welcome to the puppies page!</h1>`)
      res.end()
    }
  }
})

server.listen(3000, 'localhost')
```


We can determine how to respond using the req object.
However, this is about as far as Node abstracts things for us. And this is good - it does this so we, or other developers like us, can flexibly abstract on it even more in the way that we want.

EXPRESS

A library for request handling in Node

In node shell, we abstracted away using the built in http.request method by using the “request” package. Here, we’ll use the “express” package to abstract upon http’s other functionality.

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

const app = express()

app.listen(3000)
```


What we called our “server” object, we’ll call “app”. So far, not very different from using http.

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

const app = express()

app.get('/', (req, res, next) => {
  res.send(`<h1>Welcome to the main page</h1>`)
})

app.listen(3000)
```


…but Express starts to shine right away. Express cuts right to the core of what we want to do: serve up a certain message based on the HTTP verb and url

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

const app = express()

app.get('/', (req, res, next) => {
  res.send(`<h1>Welcome to the main page</h1>`)
})

app.get('/puppies', (req, res, next) => {
  res.send(`<h1>Welcome to the puppies page</h1>`)
})

app.listen(3000)
```


This ends up being a much easier way of describing the way our server responds to certain requests. As we explore other features of express, we'll see that it does even more work for us in its ability to parse things like query strings,

DEMO

- Remember: Chrome == client, Node == server
- Server is in “Norway”
- Install express & save it
- Create server with express (`app.listen`)
- app.get some html (use backtick strings)
    - play with setting `res.setHeader('Content-Type', 'text/plain')`
- app.get some different pages

SUMMARY

- We can use Node to easily create servers
- The `express` library makes it even easier to write (and subsequently maintain) our server code
- The `express` library is just a (light) abstraction over the built-in node `http` module, which is itself an abstraction over some code written in C++ making lower-level system calls