Integration

How do I get the application's FRONT END

(The user interface, for example, an HTML web page displayed by a browser)

To talk to the application's BACK END ?

(For example, a MySQL database)

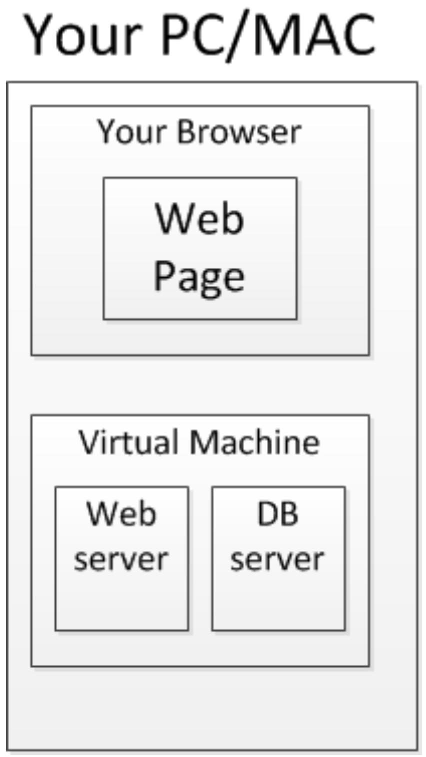
Suppose my application must

– Display a form and collect user data entry

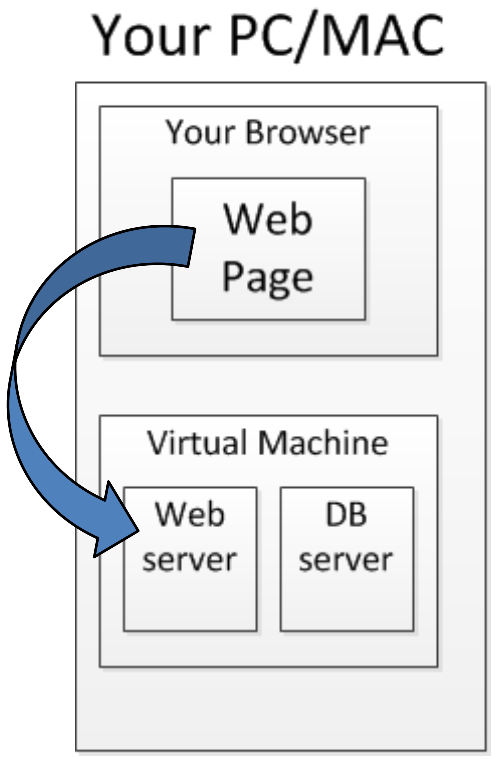
– Find and display data from the database, or

– Find and update data in the database

Integration

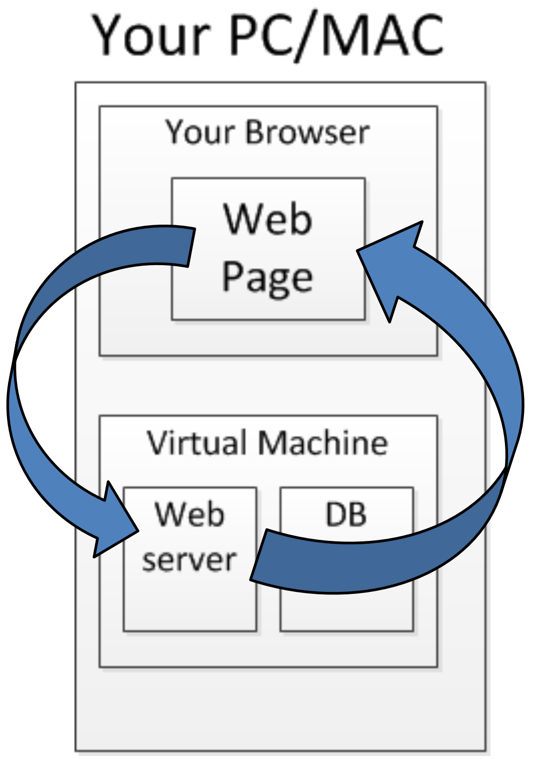


http: GET request
Passing a URL



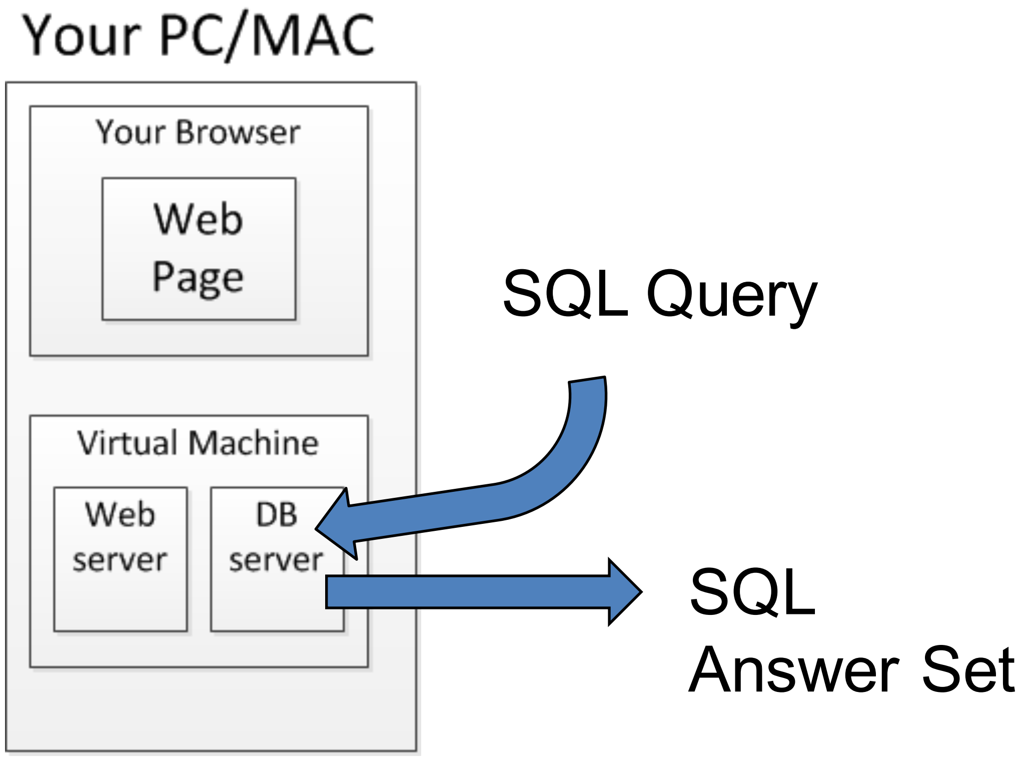
Integration

http: GET request
Passing a URL



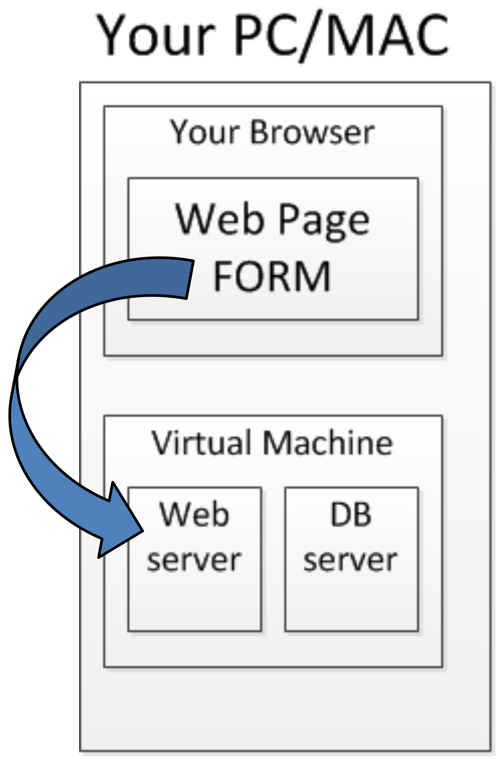
Sends back a
Web page

Integration

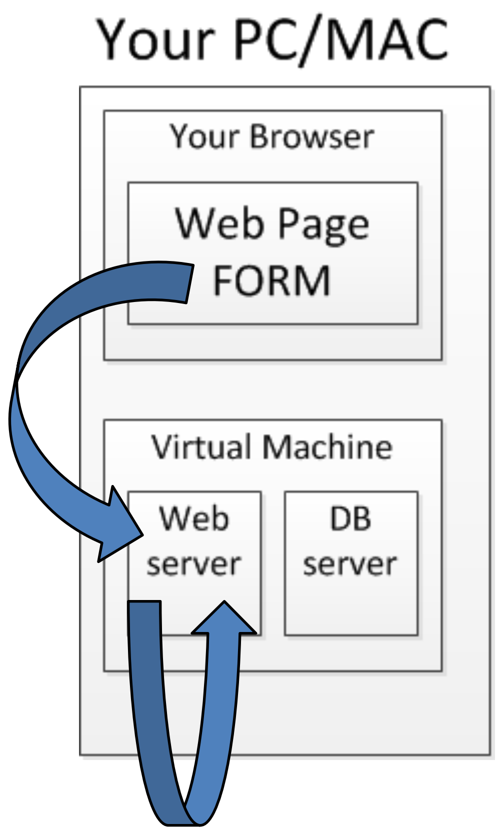


Integration

1. http: pass FORM data



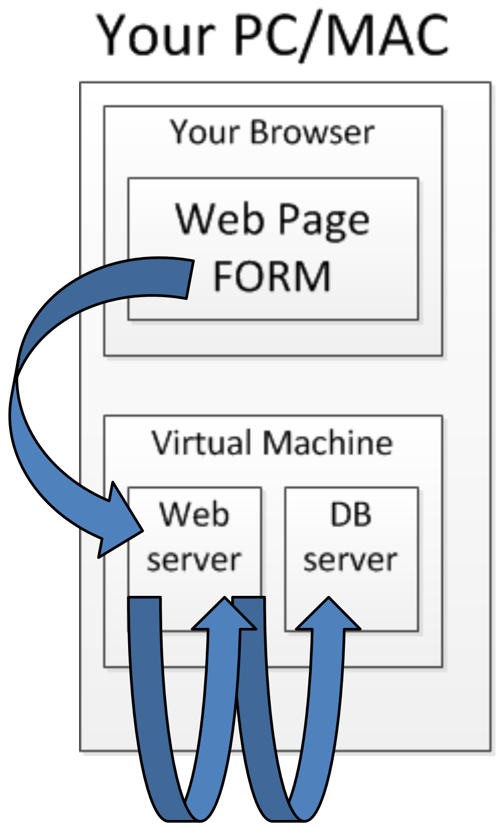
Integration



1. http: pass FORM data

2. Parse and Process FORM data

Integration



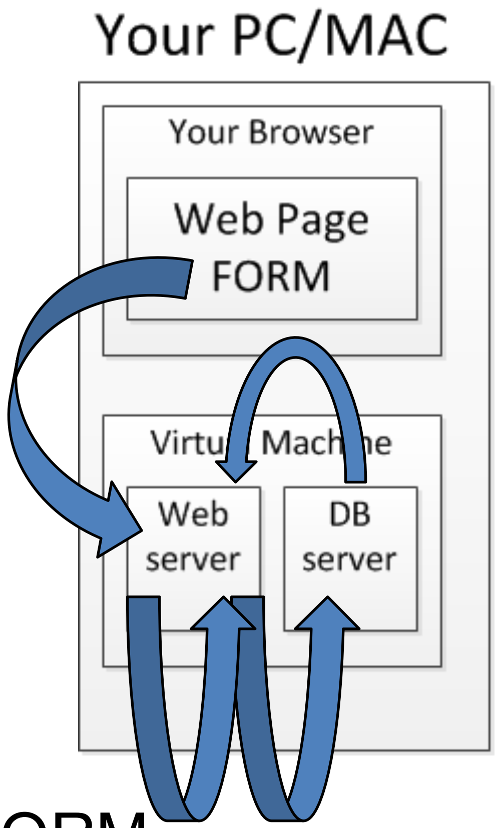
1. http: pass FORM data

2. Parse and Process FORM data

3. Connect to DB and Submit a query

Integration

1. http: pass FORM data



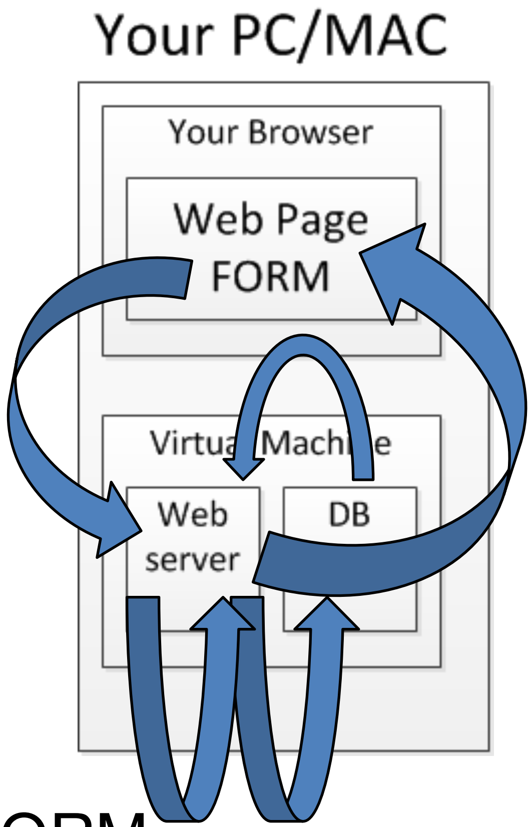
4. Return answer set

2. Process FORM data

3. Connect to DB and Submit a query

Integration

1. http: pass FORM data



5. Parse SQL answer, Build and send back a Web page

4. Return answer set

2. Process FORM data

3. Connect to DB and Submit a query

Integration

Consider the software stack we are using here.

Integration

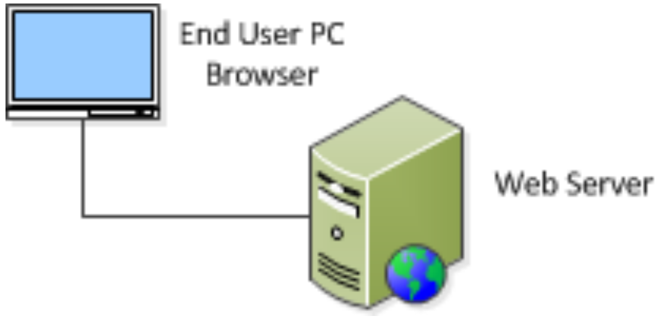
Three Components:

- **HTML** – Takes a marked up file and renders it in the browser. Runs on the PC’s browser.

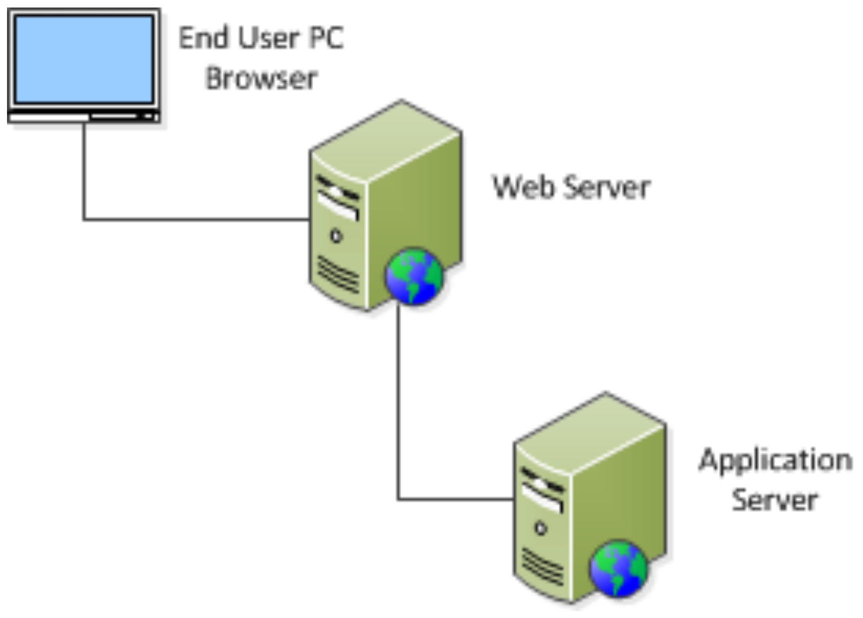
- **NodeJS** – A server-side scripting language. Runs on a web server.

- **SQL** – Communicates with the database server. The DBMS runs on a database server.

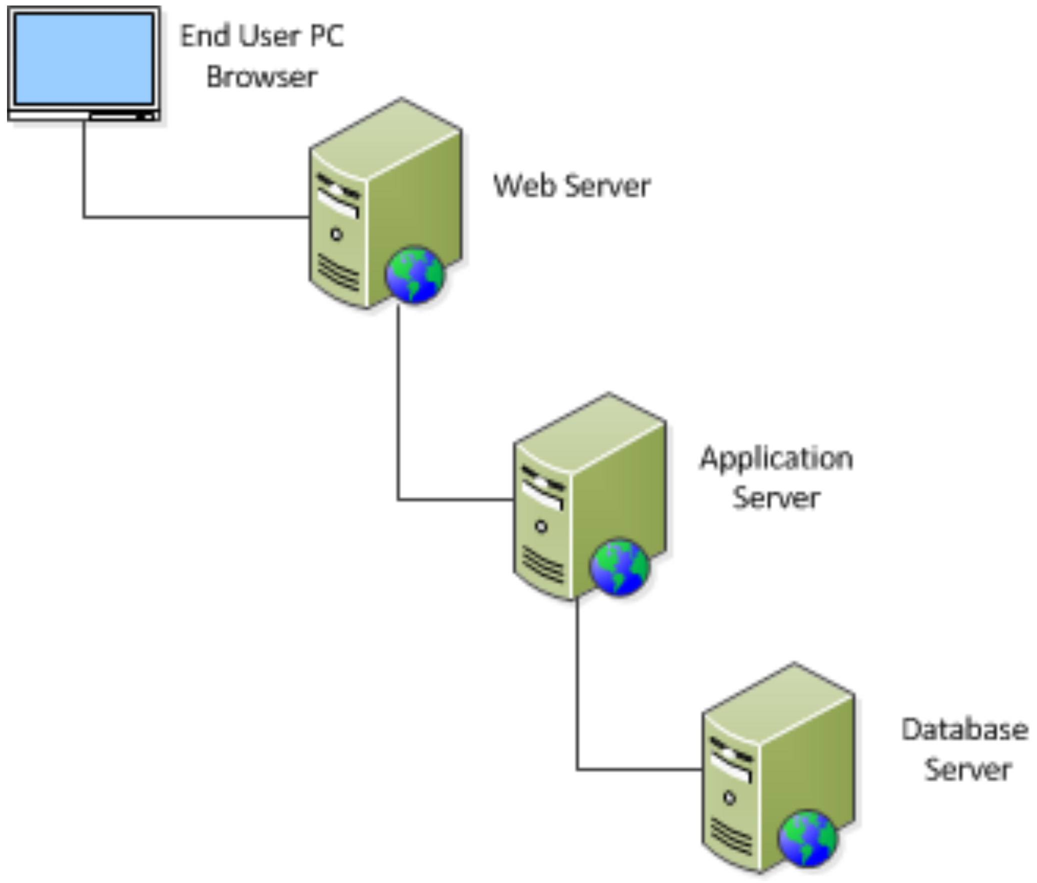
Application Architecture



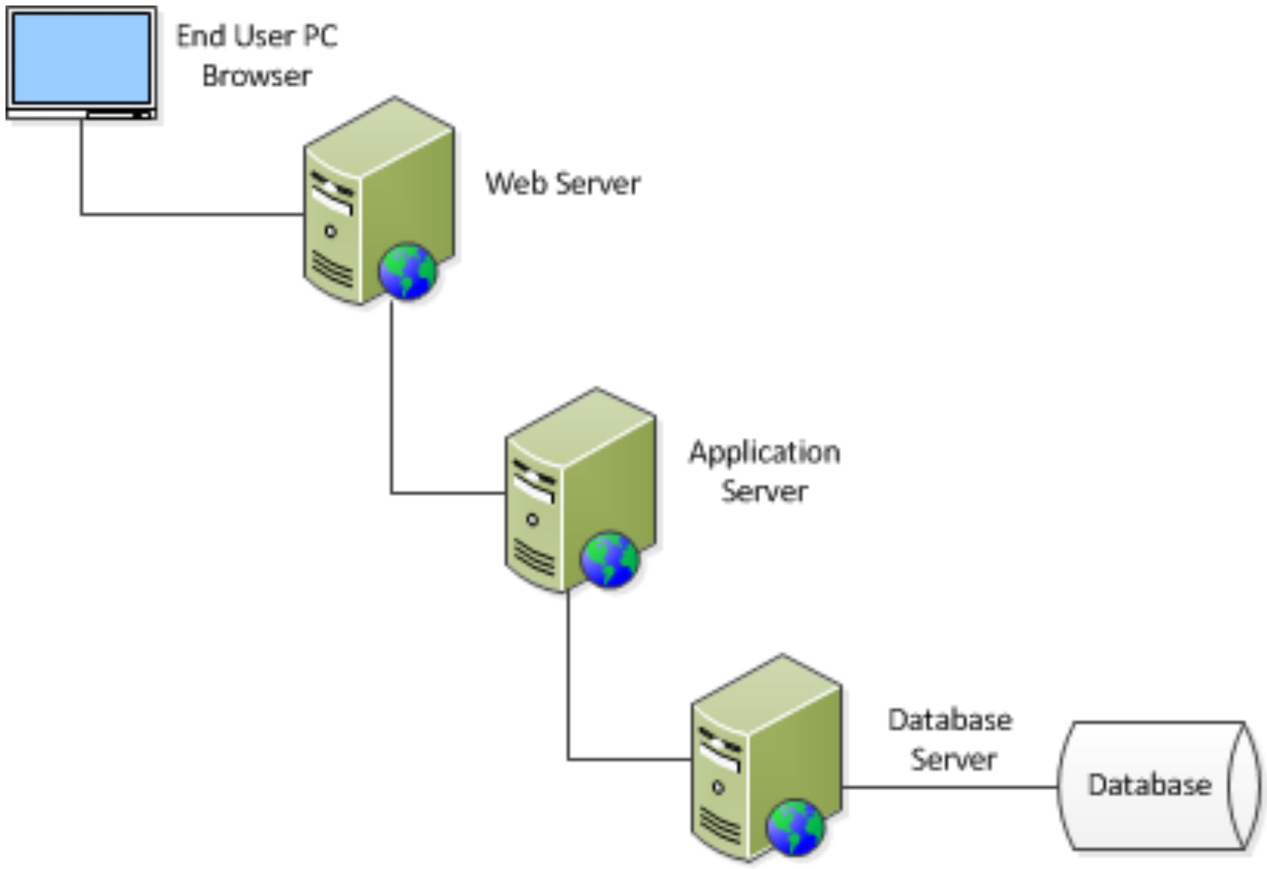
Application Architecture



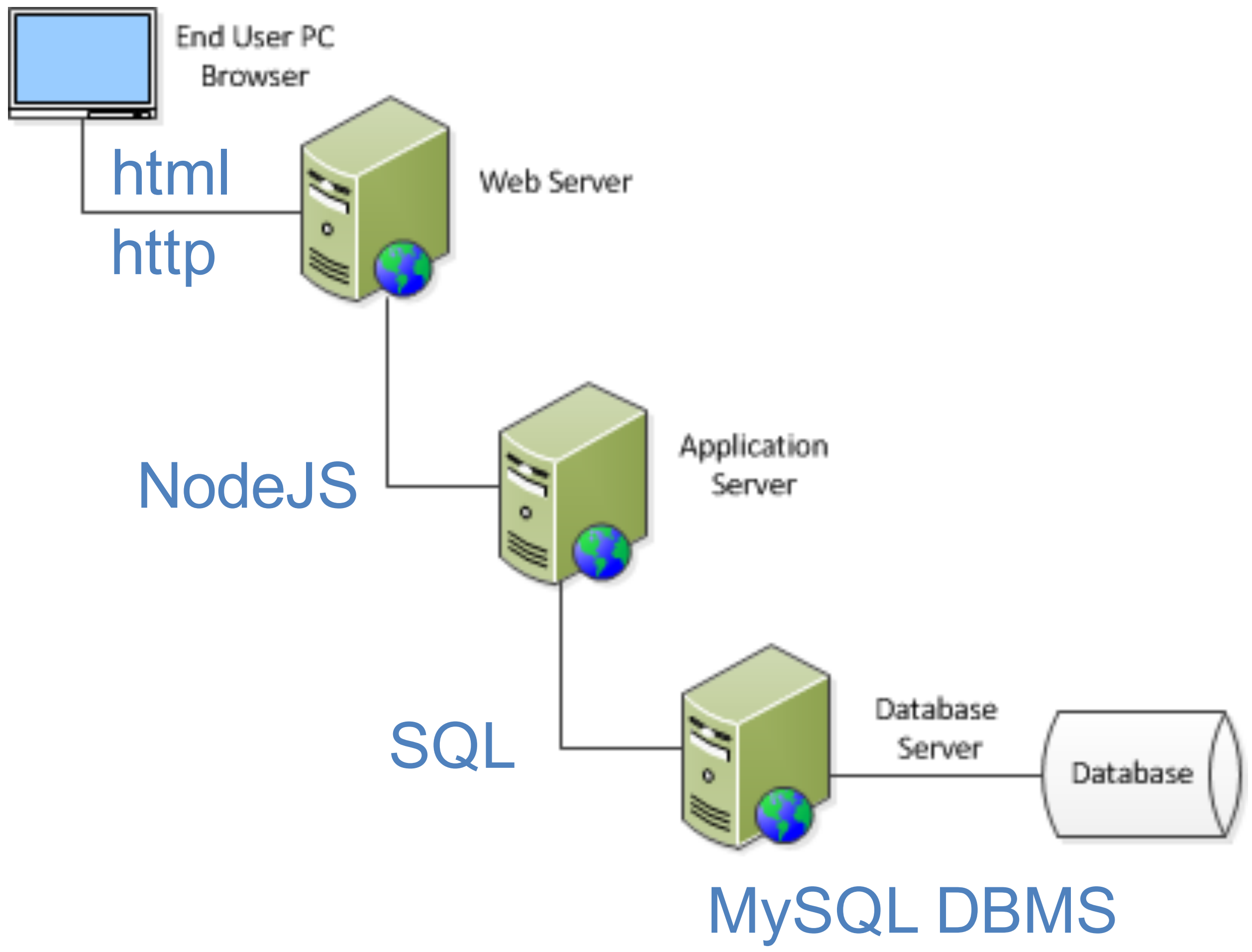
Application Architecture



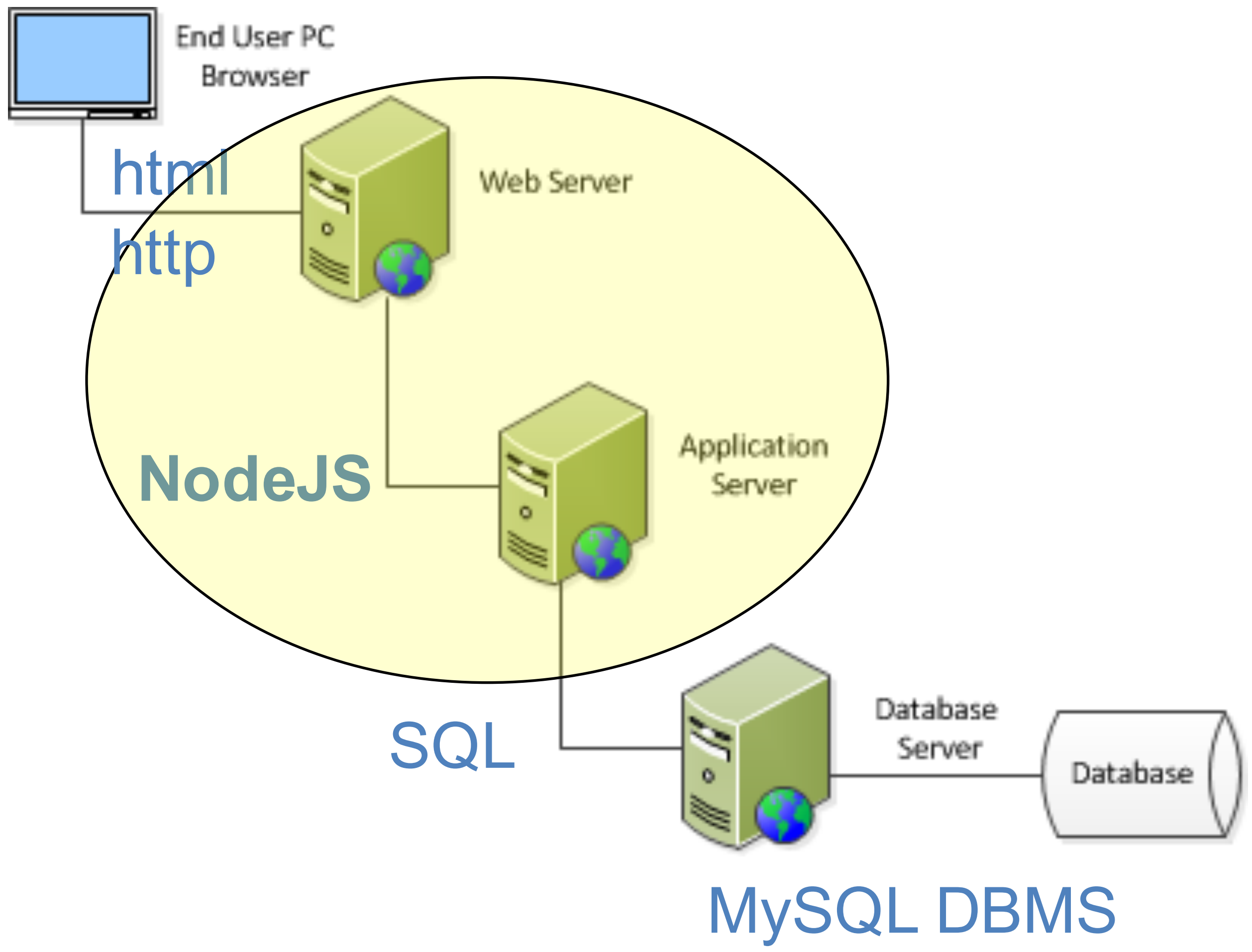
Application Architecture



Application Architecture



Application Architecture



NodeJS Basics

NodeJS is a multi-purpose server-side processing engine.

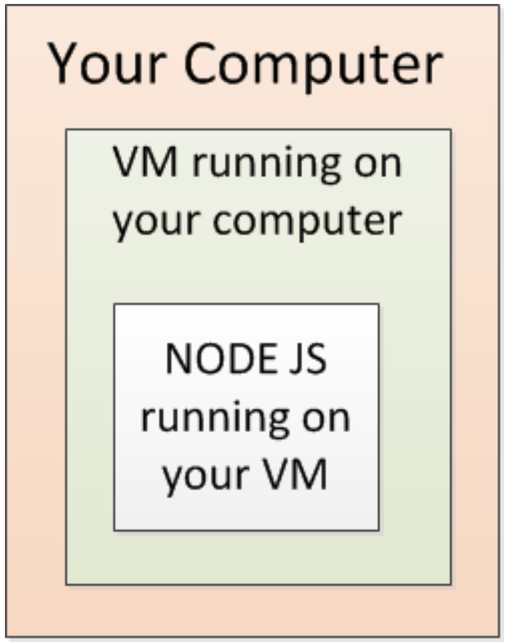
- It is Open-Source (GPL – Gnu Public License)
- It is FREE
- It runs anywhere (Windows, Linux, Unix, Mac OS X)
- It uses the Java Script programming language – the “default” language for most web-based applications.
- It looks good on your resume.

NodeJS Basics

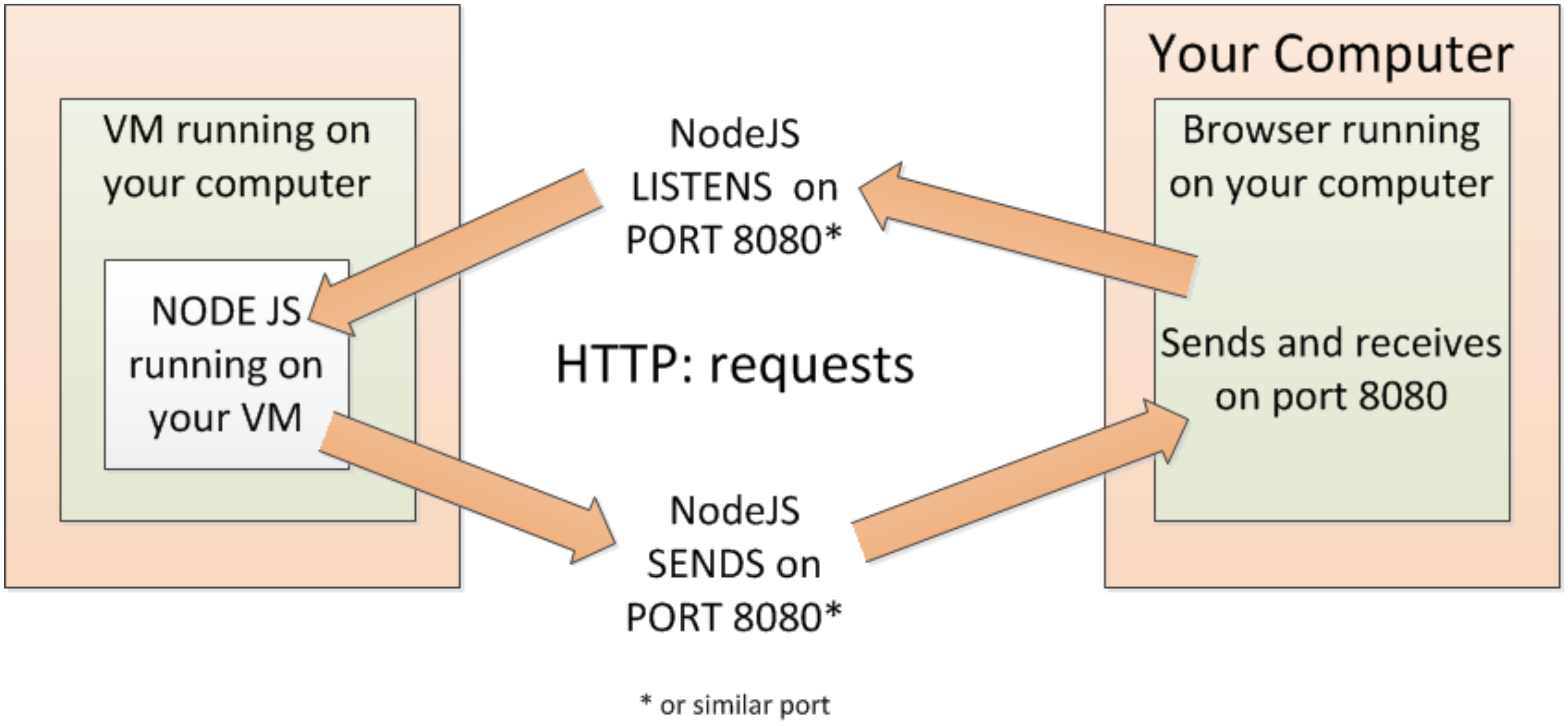
NodeJS can process HTTP requests from your browser:

- Node.js can generate dynamic page content – it *creates* the HTML on the fly
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect and process form data from an HTML page
- Node.js can read, add, change, delete data in your database

NodeJS Architecture:



NodeJS Architecture:


VM running on your computer
NODE JS running on your VM
NodeJS LISTENS on PORT 8080*
HTTP: requests
NodeJS SENDS on PORT 8080*
Your Computer
Browser running on your computer
Sends and receives on port 8080
* or similar port

NodeJS Basics

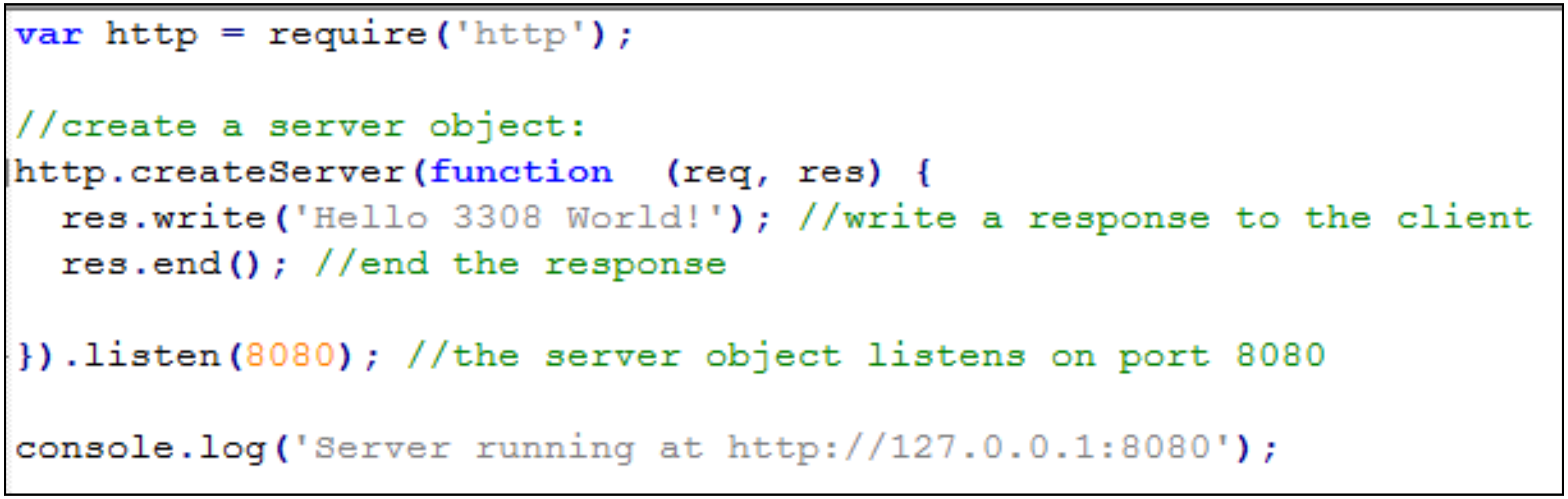
- What am I going to show you?
  - I downloaded and installed NodeJS and NPM (node package manager) and Express on my MAC
  - I use Mac terminal window to interface with the node, set up a path to the node directory
  - I will demo some basic features:
    - Starting up the node
    - Accessing Mac file systems from the node
    - Grab and display an HTML file
    - Have the node parse a URL sent from the client
    - Display different files depending on what’s passed in the URL
    - Connect to a database via NodeJS
    - Run a query and process the results via NodeJS

NodeJS Basics

- Starting up the node
  - Node is initiated from the command line
  - Node runs in the background until you stop it `(<CTRL>+C)`

  This code initiates the Node running in backround:

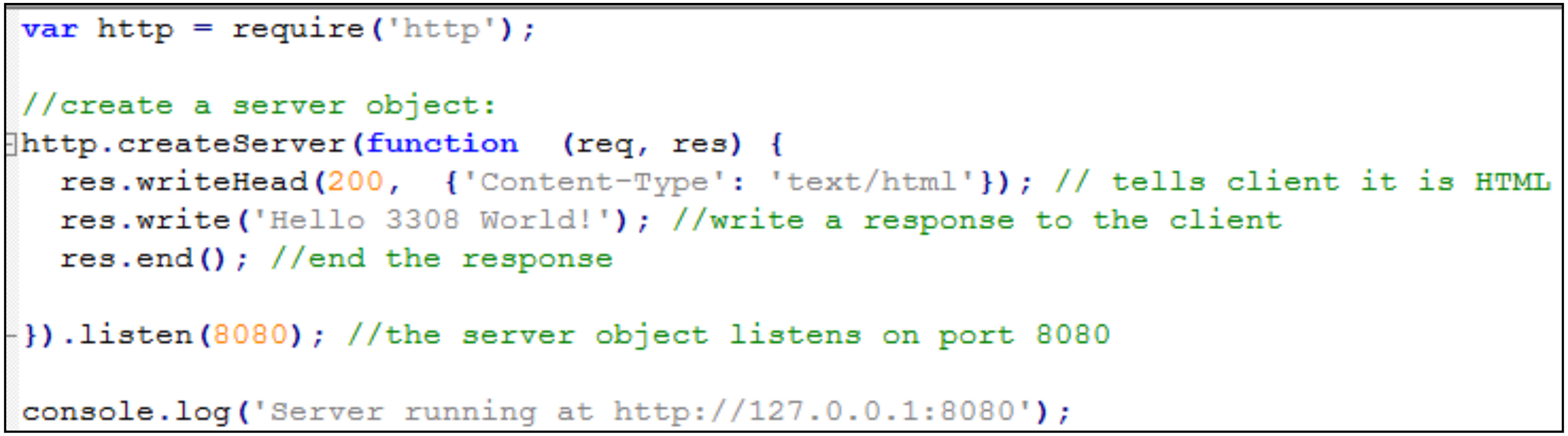
  (run `StartServer.js`)



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

//create a server object:
http.createServer(function  (req, res) {
  res.write('Hello 3308 World!'); //write a response to the client
  res.end(); //end the response

}).listen(8080); //the server object listens on port 8080

console.log('Server running at http://127.0.0.1:8080');
```

NodeJS Basics

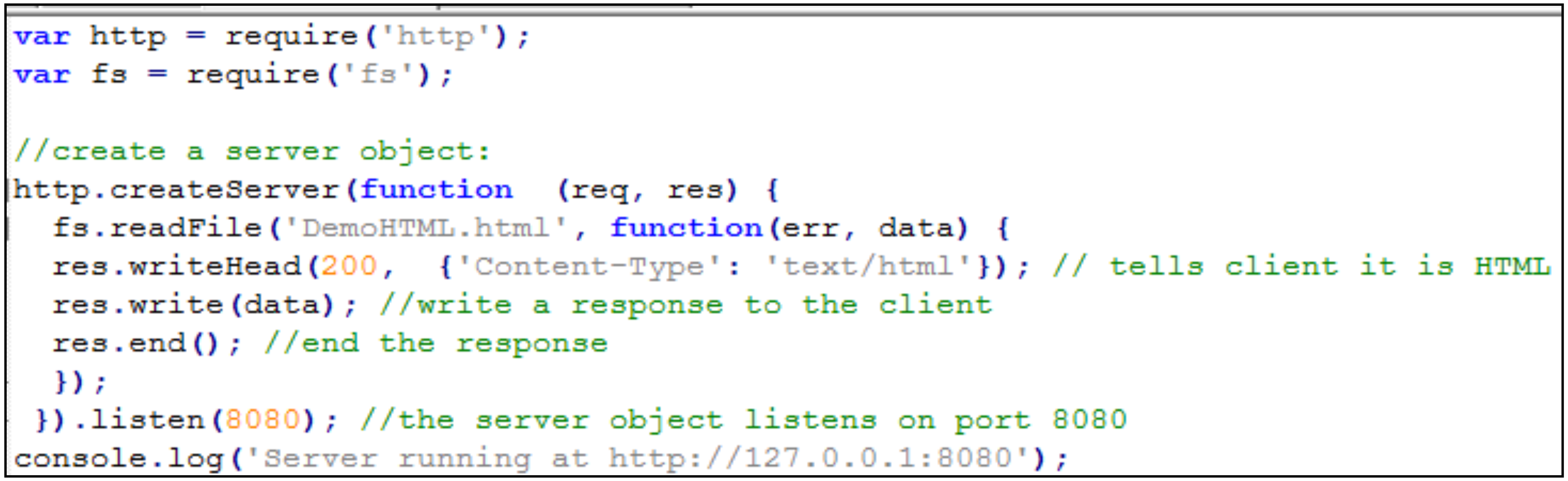
- The createServer function has two arguments:
  - "`req`" is the request coming in from the client
  - "`res`" is the result begin sent to the client
  - This code adds a `writeHead` to the client ➔ it's HTML !!



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

//create a server object:
http.createServer(function  (req, res) {
  res.writeHead(200,  {'Content-Type': 'text/html'}); // tells client it is HTML
  res.write('Hello 3308 World!'); //write a response to the client
  res.end(); //end the response

}).listen(8080); //the server object listens on port 8080

console.log('Server running at http://127.0.0.1:8080');
```

NodeJS Basics

- The fs module allows node to work with the file system on your computer
- `fs.readFile()` method allows node to read a file

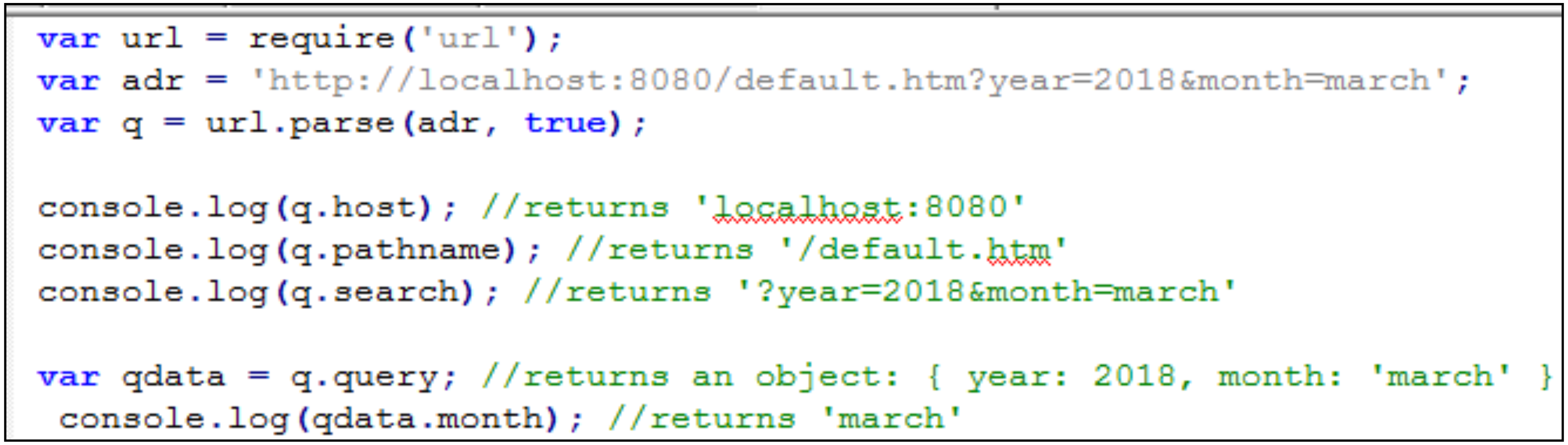


```
var http = require('http');
var fs = require('fs');

//create a server object:
http.createServer(function  (req, res) {
  fs.readFile('DemoHTML.html', function(err, data) {
  res.writeHead(200,  {'Content-Type': 'text/html'}); // tells client it is HTML
  res.write(data); //write a response to the client
  res.end(); //end the response
  });
 }).listen(8080); //the server object listens on port 8080
console.log('Server running at http://127.0.0.1:8080');
```

- Run this as `DemoHTML.js`
- It reads a file `DemoHTML.html`, passes it into “`data`”, and writes “`data`” to the result sent back to the browser.

NodeJS Basics

- The url module allows node to parse a URL passed to it
- `url.parse()` method parses out host, pathname and variable values from a URL (which will eventually be passed from the client.)



```
var url = require('url');
var adr = 'http://localhost:8080/default.htm?year=2018&month=march';
var q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2018&month=march'

var qdata = q.query; //returns an object: { year: 2018, month: 'march' }
 console.log(qdata.month); //returns 'march'
```

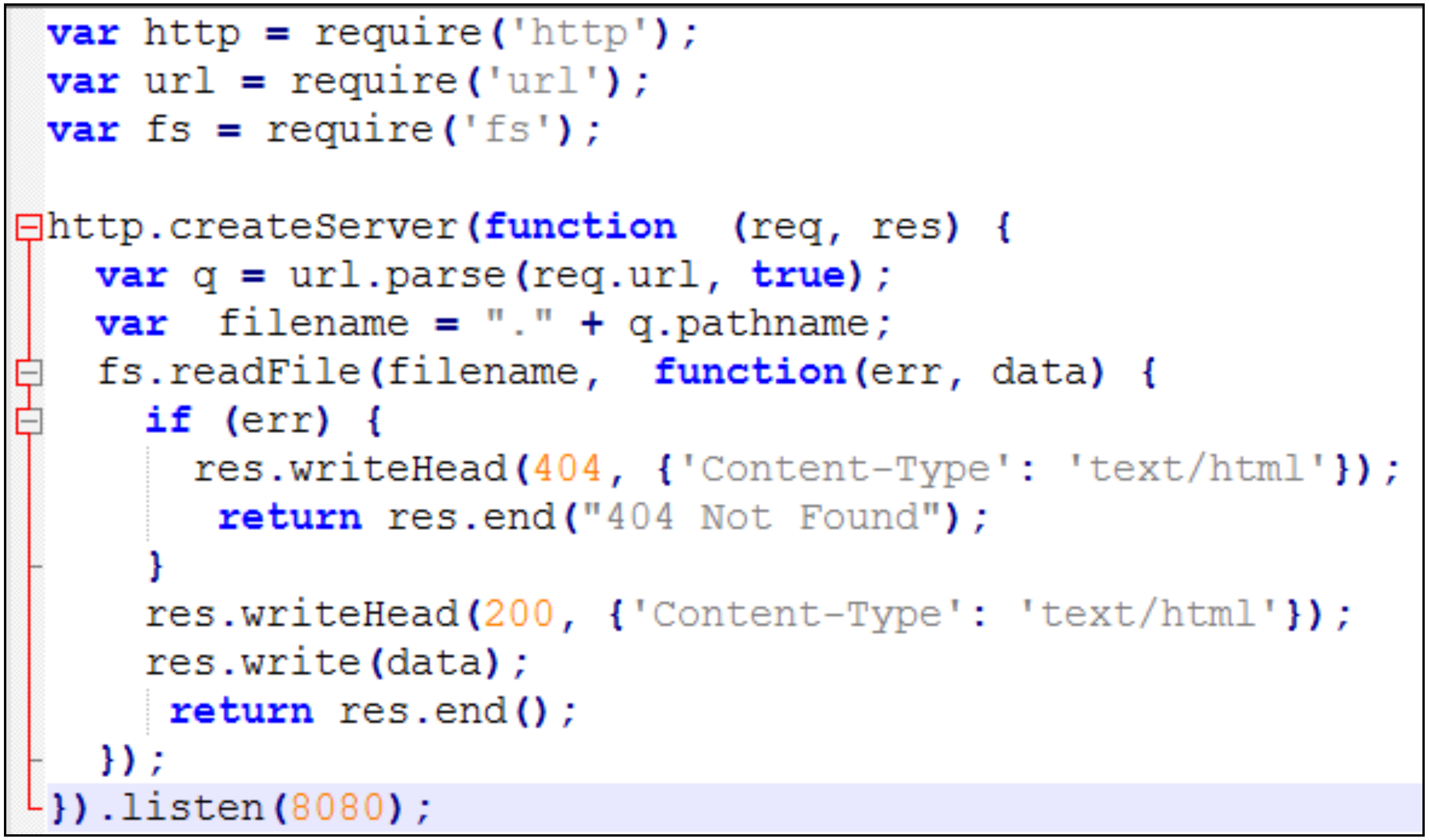
Run “`parseURL.js`” in a second command prompt !

NodeJS Basics

- Now, we can combine the URL parser with the File Reader
  - For this example, we will use two HTML files:
    - Hello.html and Goodby.html
  - We will pass from the browser the URL indicating which HTML file to write

NodeJS Basics

- This code starts the node, retrieves a URL from the browser and opens one of two files specified. Run `URLFile.js`



```
var http = require('http');
var url = require('url');
var fs = require('fs');

http.createServer(function  (req, res) {
  var q = url.parse(req.url, true);
  var  filename = "." + q.pathname;
  fs.readFile(filename,  function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
       return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
     return res.end();
  });
}).listen(8080);
```

Using HTML Forms

- HTML forms
- How are they used?
  - Use the browser’s window as a data entry screen
  - Collect information from the user
  - Pass it to the web server via http
  - Invoke a server-side script
  - Passes ***form data*** as input to the script
  - Script on server parses out the form data

Using HTML Forms

- `<form>` tag has several attributes – two are required
- ACTION
  - `<form action="http://URL">` name of a program on the web server
    - URL specifies the location of the executable file on the web server
  - `<form action="mailto:mailrecipient">` sends an email
- METHOD
  - `<form method="POST" >` or `<form method="GET">`
    - **POST** when you have large amount of data being sent, encryption available, a two-step process
    - **GET** for small amounts, no security – all in one step

    `<form enctype=`

      » multipart/form-data (default)

      » text/plain (used only for mailto)

Using HTML Forms

- <form> examples
- Text Box

```
<input type="text" name="Name" size="20" maxlength="30">
```

- Radio Button(s)

```
<input type="radio" name="Gender" value="M" /> Male
<input type="radio" name="Gender" value="F" /> Female
```

Using Forms

- Check Box(es)

```
<input type="checkbox" name= "size" value="S"
  checked="checked" />Small
<input type="checkbox" name="size" value="M" />Medium
<input type="checkbox" name="size" value="L" />Large
<input type="checkbox" name="size" value="XL" />X-Large
```

Using Forms

- List Box

```
<select name="Grade" size="3">
       <option>A
       <option>B
       <option>C
       <option>D
       <option>F
</select>
```

Using Forms

- List Box via <select> tag
  - **Size** attribute
    - When absent: you get a "drop down list", first item selected by default
    - When present: indicates the number of items in the list
  - **Selected** attribute: specifies selected item
  - **Multiple** attribute: when "yes", can click > 1

Using Forms

```
<input type="submit" />
<input type="reset" />

<textarea name="comments" cols="40" rows="8">
```

Using Forms

- Sample FORM code

```
<html>
<head>
        <title>Form Demo</title>
</head>
<body>
<form enctype="multipart/form-data"
          action="http://localhost:8080/handleform.js">

    <h2>Name:</h2>
         <input type="text" name="Name" size="20" maxlength="30" /><br><hr>

    <h2>Please Specify Gender:</h2>
        <input type="radio" name="Gender" value="M" /> Male
        <input type="radio" name="Gender" value="F" /> Female <br><hr>

    <h2>Please Select One or More Sizes:</h2>
        <input type="checkbox" name="Size" value="S" checked="checked" />Small
        <input type="checkbox" name="Size" value="M" />Medium
        <input type="checkbox" name="Size" value="L" />Large
        <input type="checkbox" name="Size" value="XL" />X-Large  <br><hr>
```

Using Forms

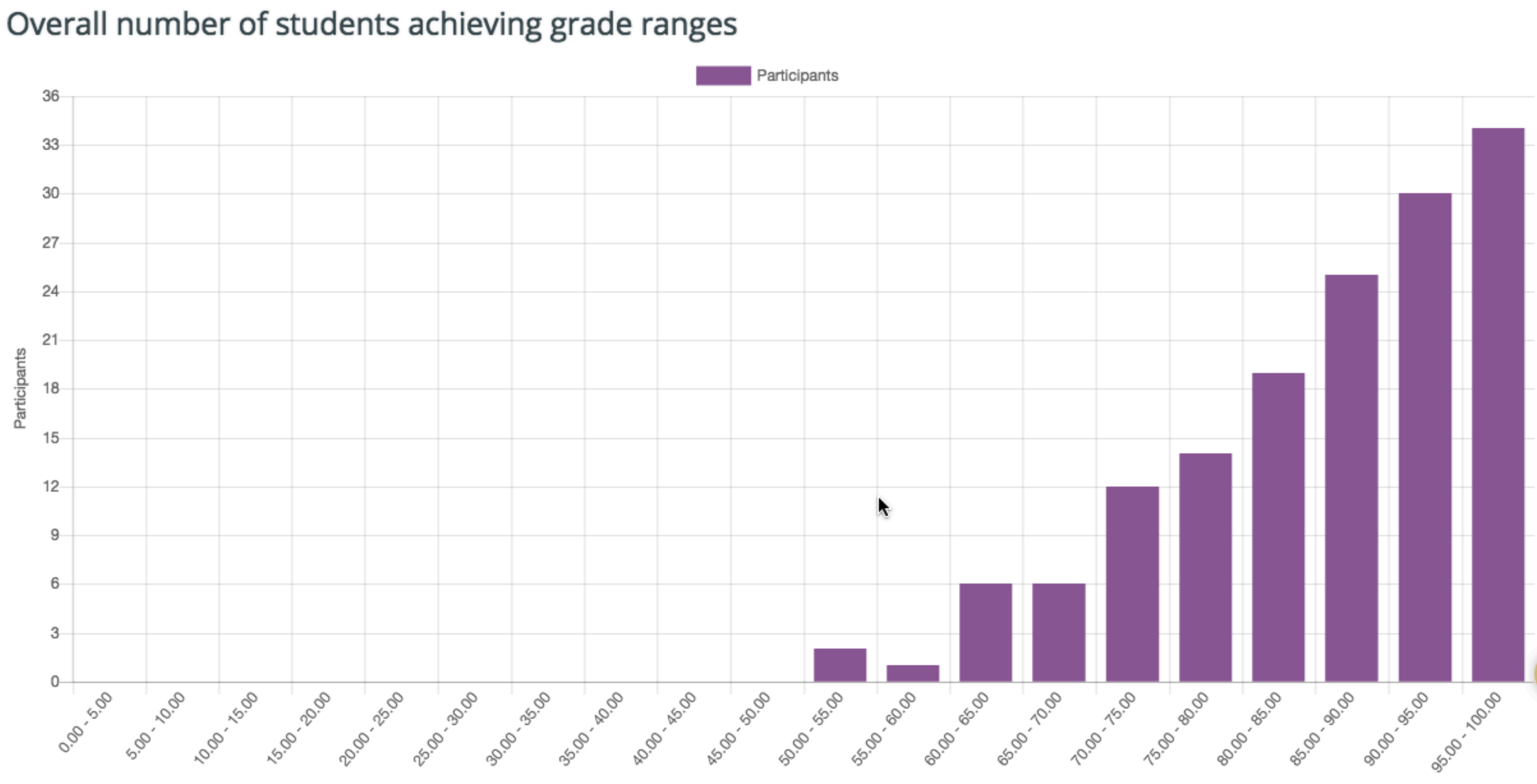
- Sample FORM code (continued)

```
<h2>Please Select Your Grade</h2>
<select name="Grade" size="5" multiple="yes" />
         <option />A
         <option />B
         <option />C
         <option />D
         <option selected="yes" />F
</select>
<br><hr><br>

Comments:<br>
<textarea name="Comments" cols="40" rows="4"></textarea>
<br><hr><br>
```

NodeJS

- Start here on Wednesday October 31

Exam Results



Average = 85.7

NodeJS Basics

- How to get NodeJS to talk to PostgreSQL (run `QueryDB_pg.js`)
  1. Build the connection string
  2. Connect (log in) to the database (`client.connect()` method)
  3. Build and run the query (`client.query()` method)
  4. View the results (`result.rows[0]`)

NodeJS Basics

- Parsing out results
  - The client.query function within JS returns a two dimensional array
    - “rows” occurs once for every row in the table, indexed by numbers starting at zero
    - “fields” occurs once for every column in the table, indexed by column name
    - We can use a “for” loop to see all the rows (run `QueryDB_2_pg.js`)

NodeJS Basics

Demonstration of Integration using a NodeJS program

`HandleForm.js` with Express framework routing

- Program Steps:
    1. Does a res.send to send the HTML form page to the browser
    2. Does a app.get to receive the HTTP GET from the form, passing back an EmployeeID when SUBMIT is pressed in the browser
    3. Builds a database connection string
    4. Connects to the database
    5. Runs a query getting the row for that EmployeeID
    6. Parse out the results
    7. Build a web page and send it back to the browser
    8. Leave the Node web server running, listening on port 8080

NodeJS Basics

The HTML Form

```
<html>
   <body>
      <form action = "http://127.0.0.1:8080/process_get" method = "GET">
         <div align=center>
          <h1>EmployeeID:</h1>
          <input type = "text" name = "employeeID">
          <br><br>
          <input type = "submit" value = "Submit">
         </div>
      </form>
    </body>
</html>
```

NodeJS Basics

The NodeJS program, first section

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

// send the form page to the browser
app.use(express.static('public'));
app.get('/index.htm', function (req, res) {
   res.sendFile( __dirname + "/" + "index.htm" );
})
```

(continued next page)

NodeJS Basics

The NodeJS program, second section

```
// process the GET request sent by the form
app.get('/process_get', function (req, res) {

// Prepare output in JSON format
   response = {
      employeeID:req.query.employeeID,
   };
   EmpID = req.query.employeeID;
   console.log(response);
```

(continued next page)

NodeJS Basics

The NodeJS program, third section

```
// Build the DB connection String
var conString = "postgres://edwardparadise@localhost/mydb";
var client = new pg.Client(conString);

// Connect to database
client.connect(function(err) {
   if(err) {
        return console.error('could not connect to postgres', err);
   }
   console.log("Connected to Northwinds Database!");
}
```

(continued next page)

NodeJS Basics

The NodeJS program, fourth section

```
// Run the query
var key = EmpID;
var queryString = 'SELECT lastname as "lastname", firstname as
"firstname" FROM nwEmployees where employeeID = ' + key + ';';

client.query(queryString, function(err, result) {
    if(err) {
       return console.error('error running query', err);
    }
console.log('result = ',result.rows[0]);
EmpLastName = result.rows[0].lastname;
EmpFirstName = result.rows[0].firstname;
```

The NodeJS program, fifth section

```
// Build and send back a web page showing the query result
res.send('</br></br><h2 align=center>Employee:</h2><h1 align=center>'+
EmpID +' - '+EmpFirstName+' '+EmpLastName+' '+'</h1>');

// Close the connection
client.end();

// Leave the NodeJS web server listening on port 8080
var server = app.listen(8080, function () {
    var port = server.address().port
    console.log("Example app listening at port", port)
})
```

NodeJS Basics

- For further information and practice:

https://www.w3schools.com/js/default.asp

https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm