Web applications technology/backend

Vittorio Zaccaria

May 10, 2018

Outline

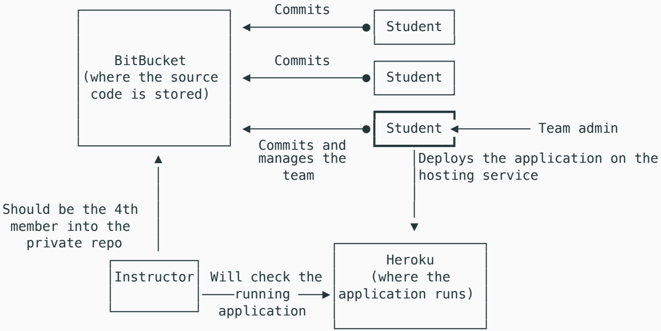
Overview

What skills you will learn

This course is fairly demanding, but is one of the most industry-applicable courses you can take. You will learn and improve the following skills:

- Programming in Javascript
- Building web services using Nodejs framework
- Experience with industry standard Web Services platform (Heroku)
- Development using Git

Course Project



BitBucket (where the source code is stored)
Commits
Student
Commits
Student
Student
Team admin
Commits and manages the team
Deploys the application on the hosting service
Should be the 4th member into the private repo
Instructor
Will check the running application
Heroku (where the application runs)

Service oriented architectures and the Web

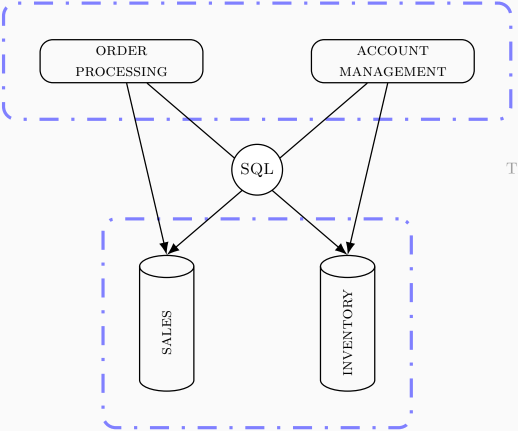
Service: definition

- Service: a *software functionality* that can be reused by different clients for different purposes.

Service oriented architecture

- Most used way to build a client/server application.
- Reuse paradigm in disguise, applications are built by integrating existing services instead of rewriting them from scratch.

No SOA

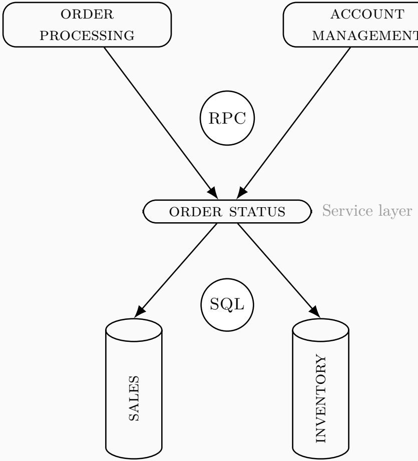


Application layer

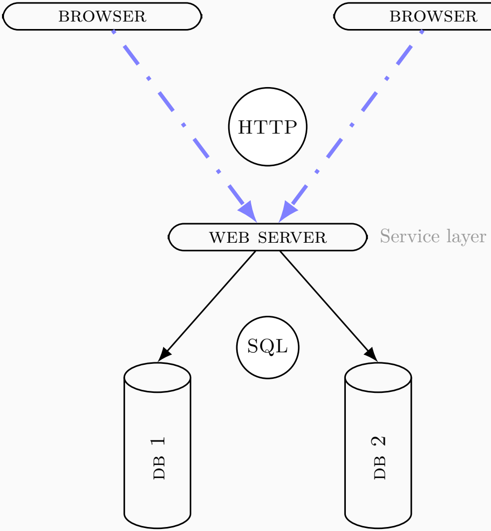
Tightly coupled communication

Databases

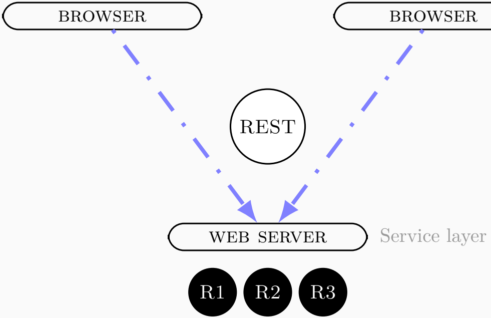
With SOA


ORDER PROCESSING
ACCOUNT MANAGEMENT
RPC
ORDER STATUS
Service layer
SQL
SALES
INVENTORY

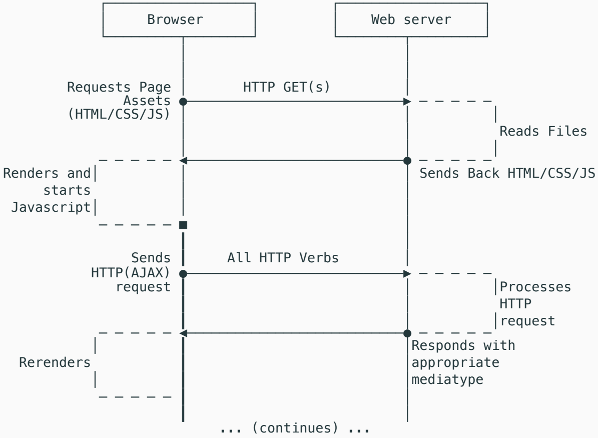
Web services



Web services

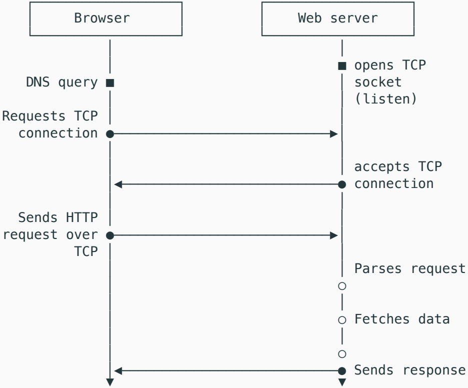


Activity pattern



HTTP-based networking and REST principles

Anatomy of a web service request



HTTP resource

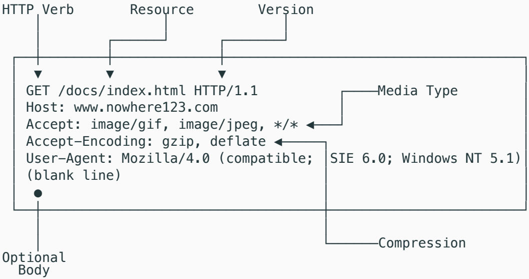
A resource has:

- an identifier (URI), i.e., a unique textual key associated with the resource:

```
scheme:[//host[:port]][/]path[?query][#fragment]
```

- a state
- a representation

HTTP request



HTTP Verb

Resource

Version

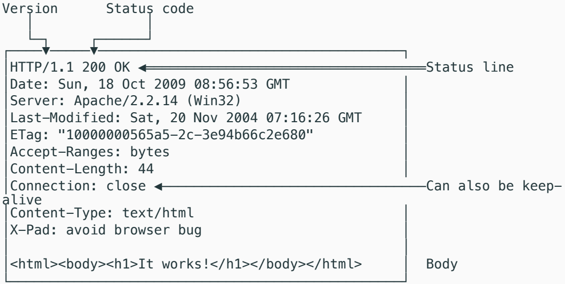
```
GET /docs/index.html HTTP/1.1
Host: www.nowhere123.com
Accept: image/gif, image/jpeg, */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

Media Type

Compression

Optional Body

HTTP response



Version

Status code

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last–Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5–2c–3e94b66c2e680"
Accept–Ranges: bytes
Content–Length: 44
Connection: close
Content–Type: text/html
X–Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

Status line

Can also be keep–
alive

Body

HTTP verbs: GET

- GOOD: `GET /fluffy_kitty.jpg`
- BAD: `GET /users/sign_out`

HTTP verbs: POST

```
POST /send-message HTTP/1.1
Host: foo.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 13

message=Hi
```

HTTP verbs: PUT

```
PUT /user/1/sign_out HTTP/1.1
Host: foo.com
```

HTTP verbs: DELETE

```
DELETE /user/1 HTTP/1.1
Host: foo.com
```

HTTP response status

The HTTP response status indicates the outcome of the request. Status codes fall into one of five categories:

- 1XX - Informational
- 2XX - Successful
- 3XX - Redirection
- 4XX - Client Error
- 5XX - Server Error

Ref: HTTP Status Code Definitions

What is REST?

- Stands for representational state transfer
- A web application that uses HTTP verbs appropriately to manipulate a resource is said compliant with the REST principle.

REST vs RPC

- RPC

```
POST /getAdUnitsByStatement HTTP/1.1
HOST: api.example.com
Content-Type: application/json

{"filter": "WHERE parentId IS NULL LIMIT 500"}
```

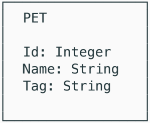
- REST

```
  single path       query
   end-point      parameters
       │                │
       │                │
       ▼                ▼
GET /ads?statement={foo}&limit=500
HOST: api.example.com
Content-type: application/json
```

Richardson maturity model

- Level 0: SOAP or XML-RPC. Single endpoint, functionality described by the request.
- Level 1: Each resource has its own URI, but requests are just GET and POST
- Level 2: Use the full power of HTTP verbs to manipulate resources
- Level 3: Hypertext as the engine of application state. Response contain hyperlinks to other URIs for performing additional actions. Example: news feeds.

REST services, a very simple example



```
PET

Id: Integer
Name: String
Tag: String
```

What is the URI of a resource?

- `/pets` - indicates a list of pets
- `/pets/12` - indicates a specific pet (n. 12)

What can I do to a resource?

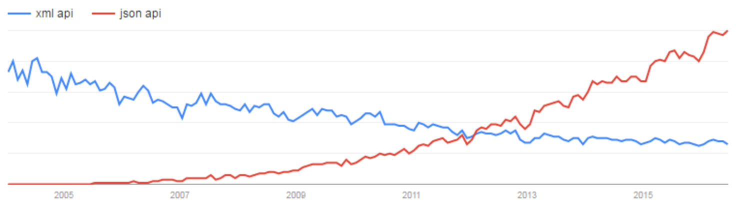
Action	Meaning	Safe	Id.
`GET /pets`	Retrieves a list of pets	Yes	Yes
`GET /pets/12`	Retrieves a specific pet	Yes	Yes
`POST /pets`	Creates a new pet	No	No
`PUT /pets/12`	Updates pet #12	No	Yes
`PATCH /pets/12`	Partially updates pet #12	No	No
`DELETE /pets/12`	Deletes pet #12	No	Yes

Example creation of a pet

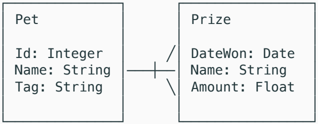
```
POST http://my.petstore.com/api/pets HTTP/1.1
Host: my.petstore.io
Content-Length: 37
Content-Type: application/json

{
  "name": "pippo",
  "tag": "dog"
}
```

JSON or XML?



What if I have a relationship between resources?



Pet

Id: Integer
Name: String
Tag: String

Prize

DateWon: Date
Name: String
Amount: Float

Relationships between resources

HTTP Action and Resource URI	Meaning
`GET /pets/12/prizes`	Retrieves list of prizes for pet #12
`GET /pets/12/prizes/5`	Retrieves prize #5 for pet #12
`POST /pets/12/prizes`	Creates a new prize in pet #12
`PUT /pets/12/prizes/5`	Updates prize #5 for pet #12
`PATCH /pets/12/prizes/5`	Partially updates prize #5 for pet #12
`DELETE /pets/12/prizes/5`	Deletes prize #5 for pet #12

Result filtering/sorting and searching

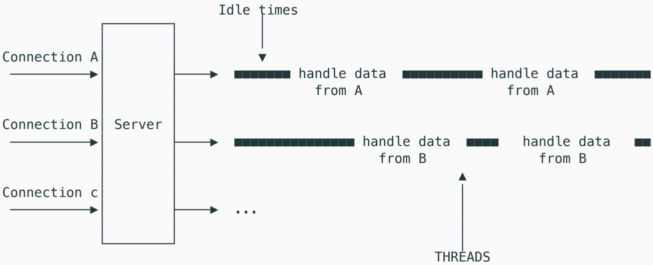
- filtering pets: `GET /pets?tag=dog`
- sorting pets by descending alphabetic order: `GET /pets?sort=-name`
- search for keyword: `GET /pets?q=miao`

Rudiments of server-side programming

Blocking I/O

```
// blocks the thread until the data is available
data = socket.read();
// data is available
process(data);
```

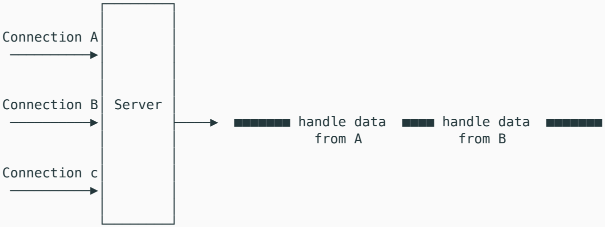
Blocking I/O in Web Servers



Non-Blocking I/O

- Most modern operating systems support another mechanism to answer incoming IO requests
- It is called non-blocking I/O
- Managed by a Synchronous event demultiplexer or Event notification interface

Non-Blocking I/O in Web Servers



Non-Blocking Engine

- Each operating system has its own interface for the Event Demultiplexer:
  - epoll on Linux.
  - kqueue on Mac OS X.
  - I/O Completion Port API (IOCP) on Windows.

NodeJS

- Node.js [1]is a platform built on Chrome V8 JavaScript engine for easily building non-blocking IO applications.

[1] We will use Nodejs 7.5.0 with all the latest ES6 goodies.

Applied server-side programming

Running NodeJS Programs

- Using the REPL. Type `node` in your command line.

```
$> node
> console.log('hello nodejs!');
hello nodejs!
undefined
>
```

- Create a .js file and type: `node file.js`

```
$> node hello.js
hello nodejs!
```

Your first Javascript program

```javascript
let x = 3;
console.log(`| Il valore della variabile e' ${x}`);
```

Higher order functions

This is probably something you've never heard of. Javascript has full-fledged higher order functions, i.e., you can use/write a function that does at least one of the following:

- takes one or more functions as arguments (i.e., procedural parameters),
- returns a function as its result.

Your first use of a higher order function

This function schedules the invocation of `func()` every `ms` milliseconds:

```
setInterval(func, ms);
```

Your first use of a higher order function

- A function without a name is called anonymous function.
- In this case, the anonymous function is also a closure, i.e., it remembers the environment in which it has been defined (variable x):

```
let x = 1;
setInterval(
  function() {
    console.log("Value of x=" + x);
    x++;
  },
  1000
);
```

Why higher order functions are important?

- The majority of APIs you are going to use are asynchronous and accept a callback (higher order function) to manage the result (or the error).
- Let’s see an example of this which is important for this course.

Your first NodeJS Very Basic Web Server

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

http.createServer(function (request, response) {
    response.writeHead(200, {
        'Context-Type': 'text/plain',
    });
    response.end('Hello World!');
}).listen(3000);

console.log('Server running at http://localhost:3000');
```

Congratulations, you have your first web server running, using Node's HTTP API!

Things you need to know

- Javascript - lexical structure
- Javascript - variables and types
    - Especially: arrays and objects
- Javascript operators and control structures

Modules

- Libraries of pre-built functions you can download and use in your own program.
- The main mechanism to enforce information hiding by keeping private all the functions and variables that are not explicitly marked to be exported.
- Node.js follows the CommonJS module system, and the builtin `require()` function is the way to include modules that exist in separate files/folders.
- Everything inside a module is private unless it is assigned to the `module.exports` variable.

Module example

In a file called `say.js`:

```
module.exports = function (msg) {
    console.log('Say: ' + mgs);
}
```

In a file called `app.js`:

```
let say = require('./say.js');
say('hello!');
```

Execute using: `node app.js`

Node Package Manager

- A package manager for Javascript Developers.
- Runs through the command line and manages dependencies for an application.
- The best way to manage locally installed npm packages is to create a `package.json` file.

Node Package Manager

- to install dependencies defined in a pacakge.json file

```
> npm install
```

- to install lodash and save the dependency in the package.json file

```
> npm install lodash --save
```