Node.JS Web Server

MSc. Tran Thi Bich Hanh – FIT HCMUS

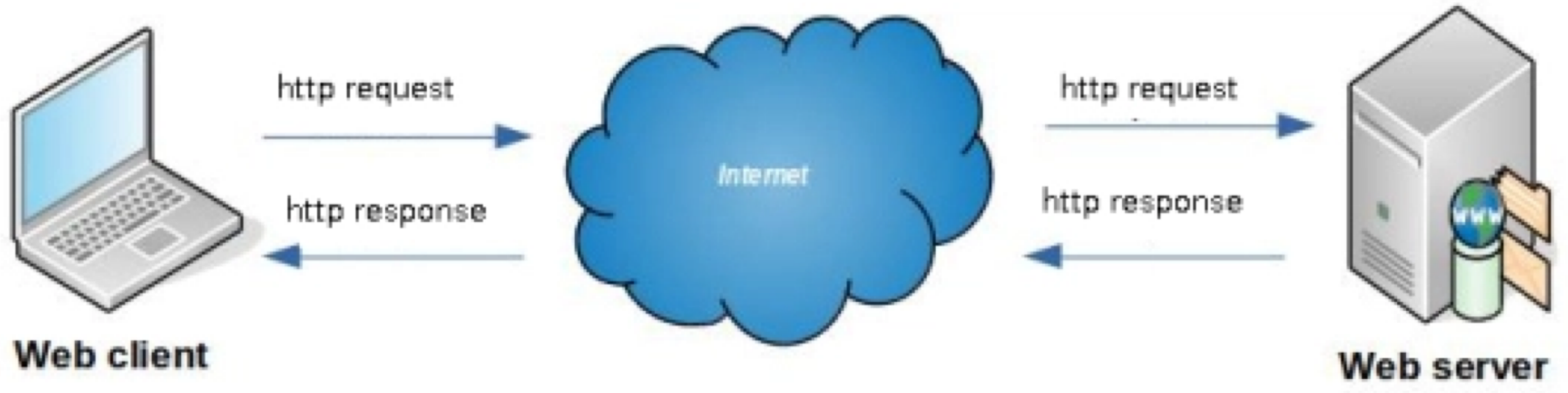
KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

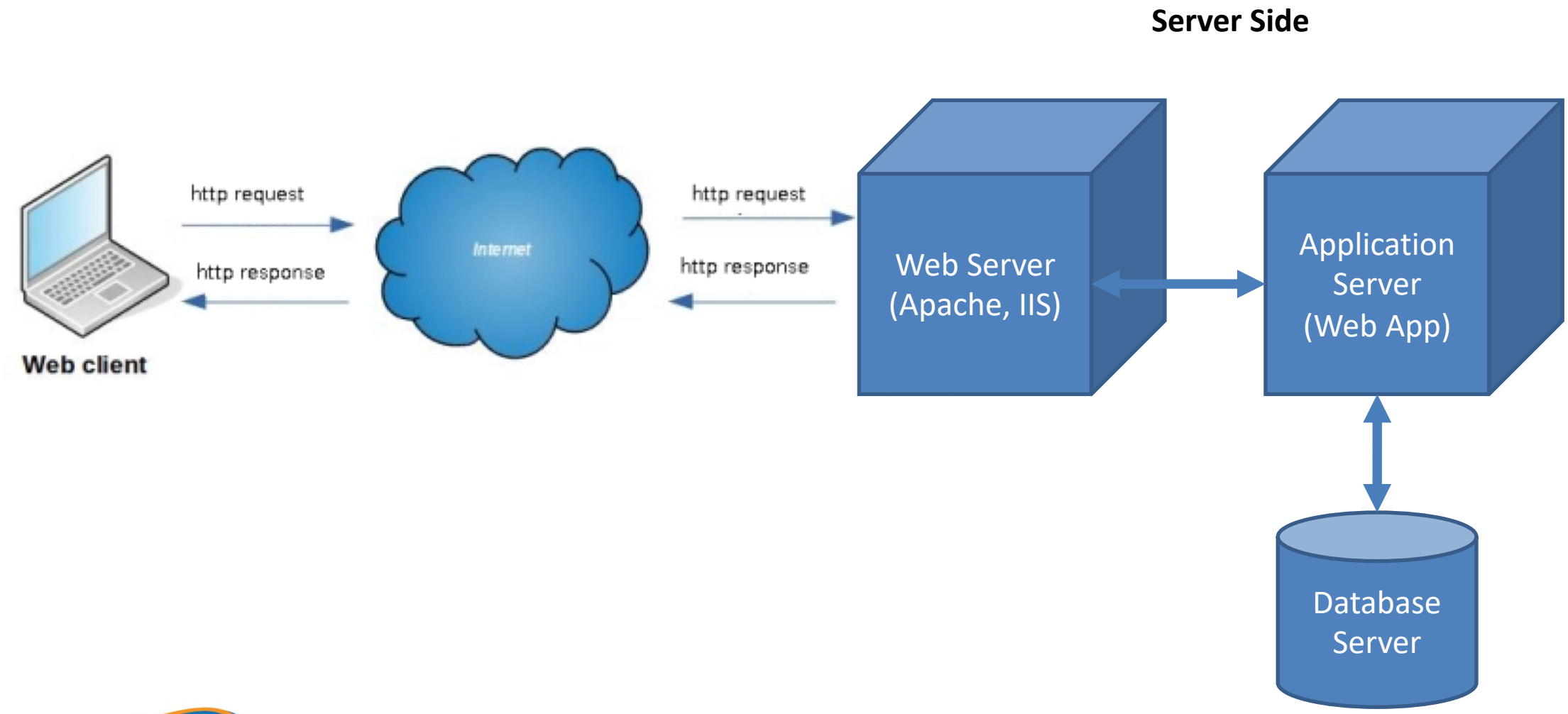
Agenda

1. Client & Server
    1. Simple NodeJS Server
    2. Request & Response
    3. Routing
2. Express JS
    1. Express Routing
    2. Middleware & Static Files
    3. View Engine
    4. Query String
    5. POST request
    6. …

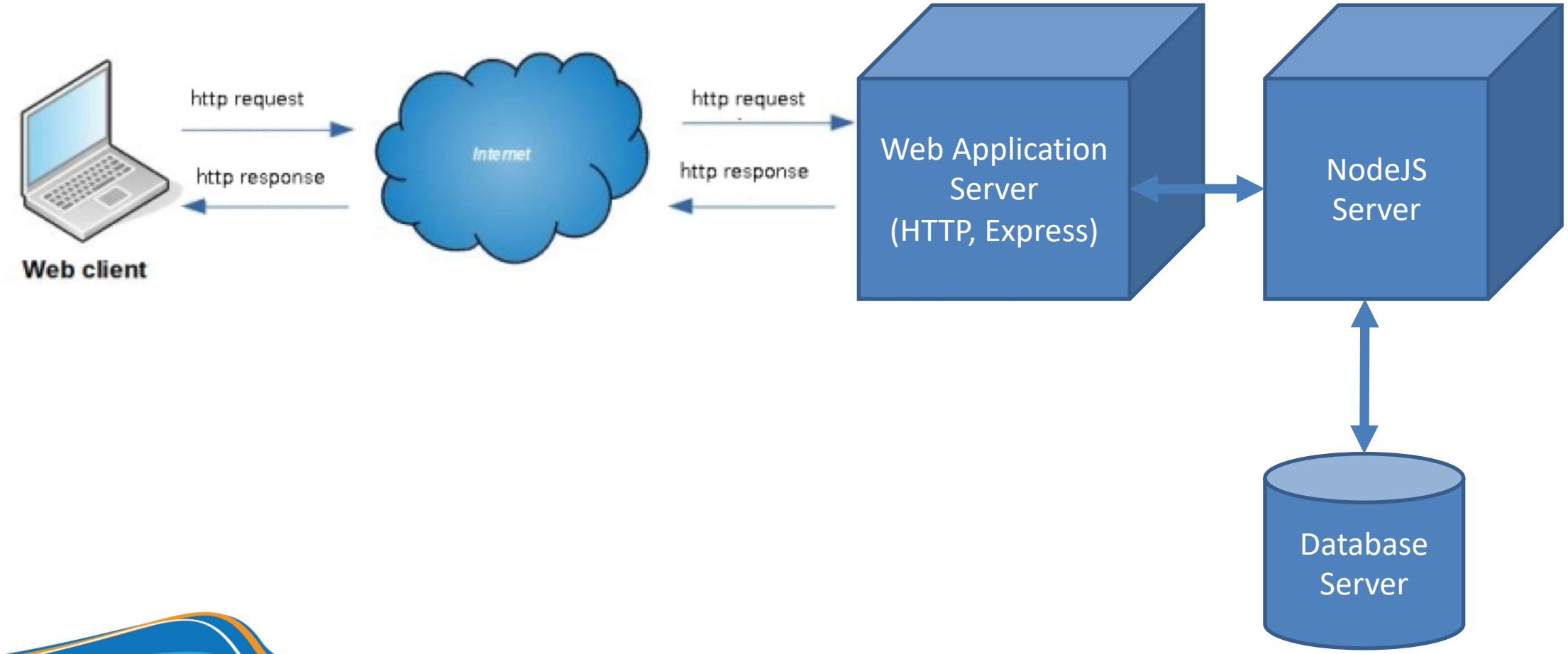
1

Client & Server


http request
http response
Internet
http request
http response
Web client
Web server


Server Side
http request
http response
Internet
http request
http response
Web client
Web Server
(Apache, IIS)
Application
Server
(Web App)
Database
Server

Server Side


http request
http response
Internet
http request
http response
Web client
Web Application Server (HTTP, Express)
NodeJS Server
Database Server

2

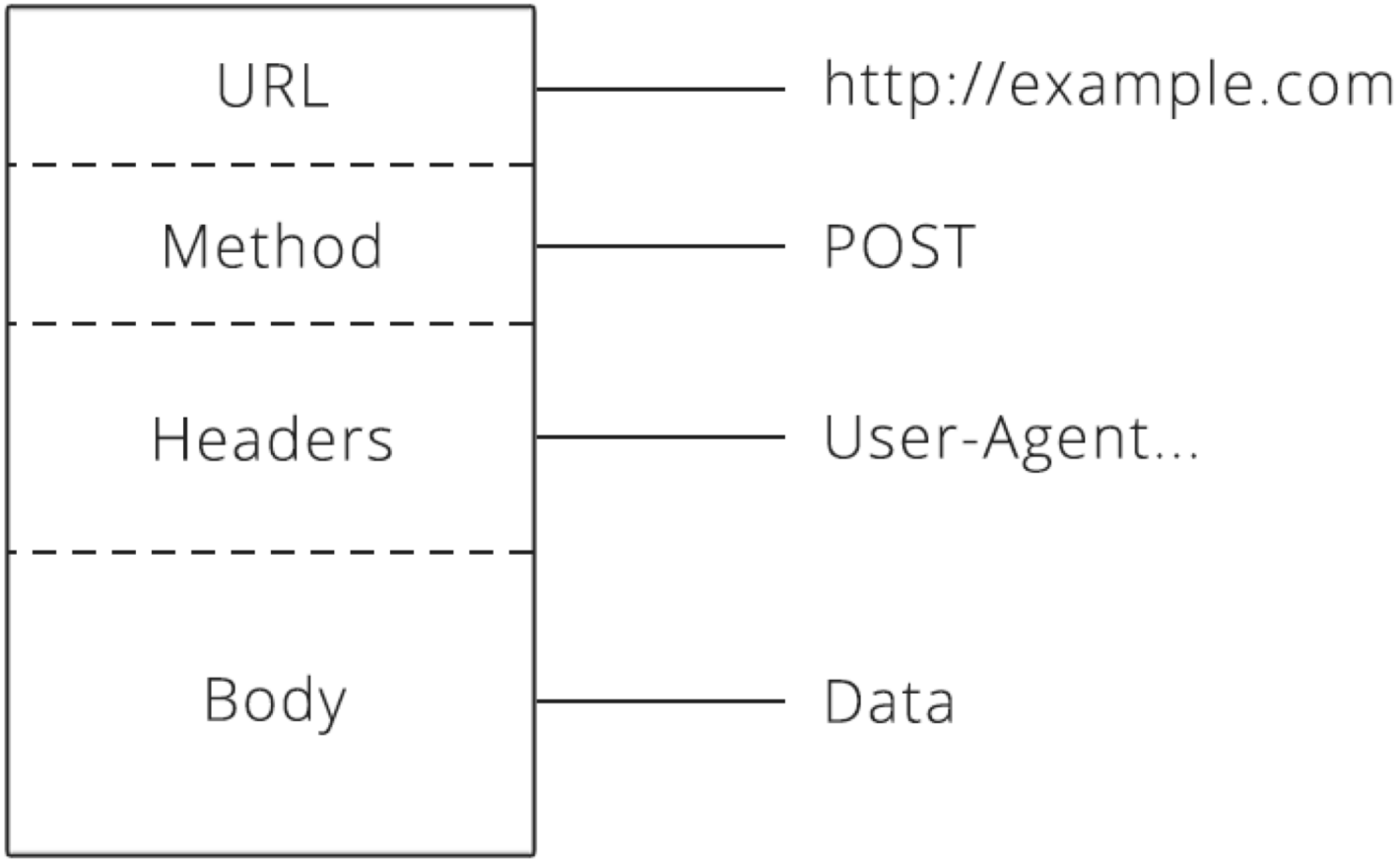
Create Web Server

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

const server = http.createServer(function(request, response) {
    response.end("Hello world from NodeJS HTTP Server");
});

server.listen(8081, function() {
    console.log('HTTP Server is listening on port 8081...');
});
```

3

HTTP Request


Request
GET /index.html HTTP/1.1
Response
HTTP/1.1 200 OK
Web Browser
Web Server


URL
http://example.com
Method
POST
Headers
User-Agent...
Body
Data

Request

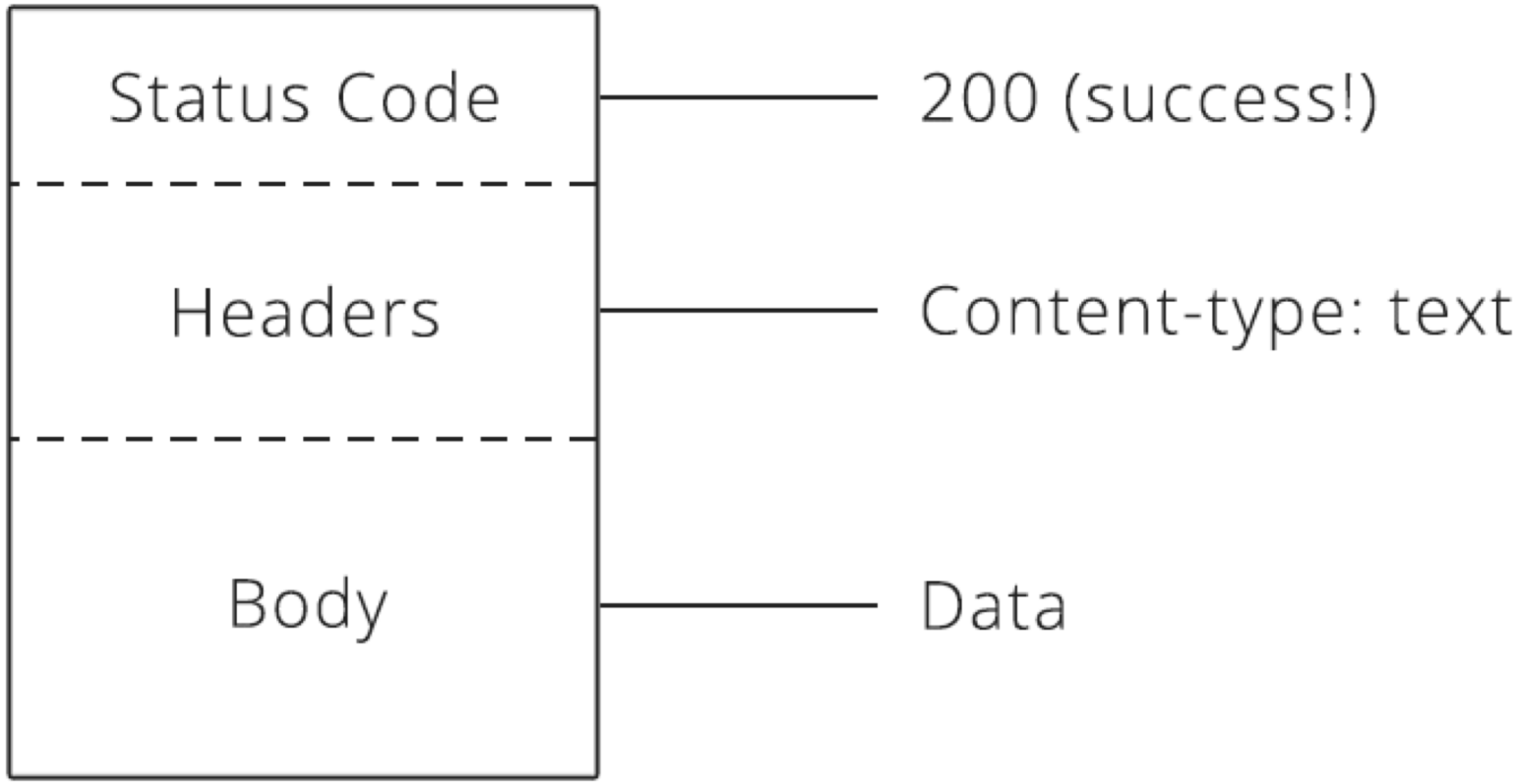
HTTP Method

Create	•HTTP POST
Read	•HTTP GET
Update	•HTTP PUT
Delete	•HTTP DELETE

4

HTTP Response


Request
GET /index.html HTTP/1.1
Response
HTTP/1.1 200 OK
Web Browser
Web Server


Status Code
200 (success!)
Headers
Content-type: text
Body
Data

Response

HTTP Status Code

Code	Ý nghĩa	Ví dụ
1xx	Information	102: Processing
2xx	Success	200: OK 201: Created
3xx	Redirection	301: Moved Permanently
4xx	Client Error	401: Unauthorized request 403: Forbidden 404: Not Found
5xx	Server Error	500: Internal Server Error

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Content-type (MIME Type)

Type	Ví dụ
text	text/plain, text/html, text/css, text/javascript
image	image/gif, image/png, image/jpeg, image/bmp…
audio	audio/midi, audio/mpeg, audio/wav…
video	video/webm, video/ogg
application	application/xml, application/json, application/pdf…

https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types

5

Node Package Manager

Node Package Manager - npm

- Init package.json
  - `$ npm init`
- Install 1 package
  - `$ npm install [-g | -s] packageName`
- Uninstall 1 package
  - `$ npm uninstall [-g | -s] packageName`
- npm packages
  - https://www.npmjs.com/

6

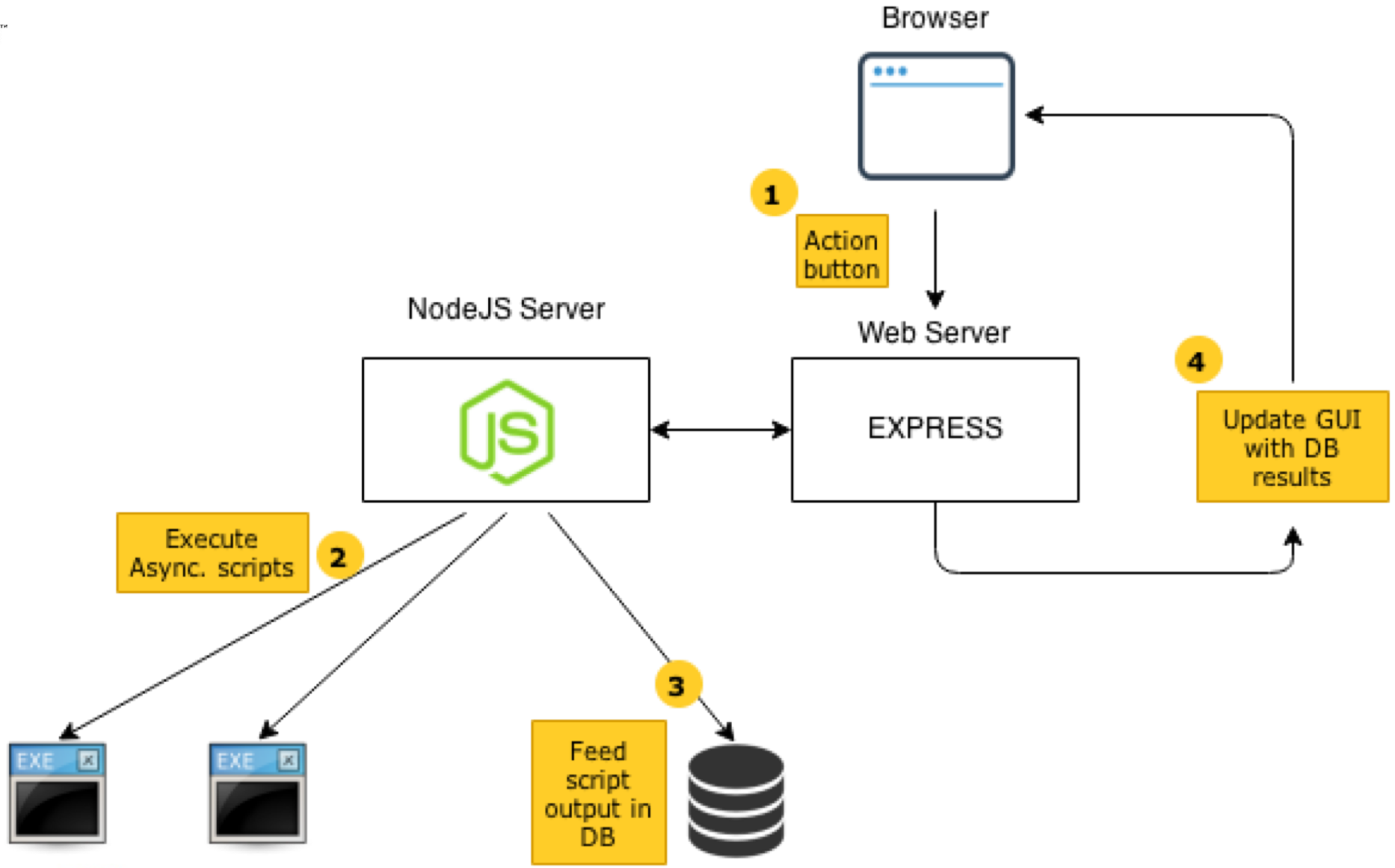
Nodemon

Node Monitor - nodemon

- Install nodemon
  - $ `npm install [-g | -s] nodemon`
- Start nodejs application using nodemon
  - $ `nodemon appFile`

7

Express JS


Browser
1
Action button
NodeJS Server
Web Server
EXPRESS
4
Update GUI with DB results
Execute Async. scripts
2
EXE
EXE
3
Feed script output in DB

ExpressJS

JS index.js ●

```
1   var express = require('express');
2   var app = express();
3
4   app.get('/', function(req, res) {
5       res.send("Hello from Express JS");
6   });
7
8   app.listen(5000, function() {
9       console.log('Server is listening on port 5000...');
10  });
```

Request Methods

- POST ➔
  - `app.post('route', function(req, res){…});`
- GET ➔
  - `app.get('route', function(req, res){…});`
- PUT ➔
  - `app.put('route', function(req, res){…});`
- DELETE ➔
  - `app.delete('route', function(req, res){…});`

Response Methods

Method	Description
res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send a response of various types.
res.sendFile()	Send a file as an octet stream.
res.sendStatus()	Set the response status code and send its string representation as the response body.

8

Express Routing

Request Methods

- POST ➔
  - `app.post('route', function(req, res){…});`
- GET ➔
  - `app.get('route', function(req, res){…});`
- PUT ➔
  - `app.put('route', function(req, res){…});`
- DELETE ➔
  - `app.delete('route', function(req, res){…});`

Route Paths

```
app.get('route', function(req, res){…});
```

- route: string, pattern or regular expression
- Special charactors: ?, +, *, () must put in ([and])
  - "/data/$book" ➔ "/data/([$])book".
- Express Route Tester
  - http://forbeslindesay.github.io/express-route-tester/

Route Paths

```
// Route: ab?cd => URL: abcd, acd
// Route: ab+cd => URL: abcd, abbbcd
// Route: ab*cd => URL: abcd, abRANDOMcd
// Route: a(bc)?d => URL: abcd, ad
app.get('/ab?cd', function(req, res) {
    res.send('/ab?cd');
});

// URL: /data/$book
app.get('/data/([$])book', function(req, res) {
    res.send('/data/$book');
});
```

9

Express Route Params

Route Parameters

```
app.get('/users/:userId/books/:bookId', function (req, res) {
  res.send(req.params)
})
```

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

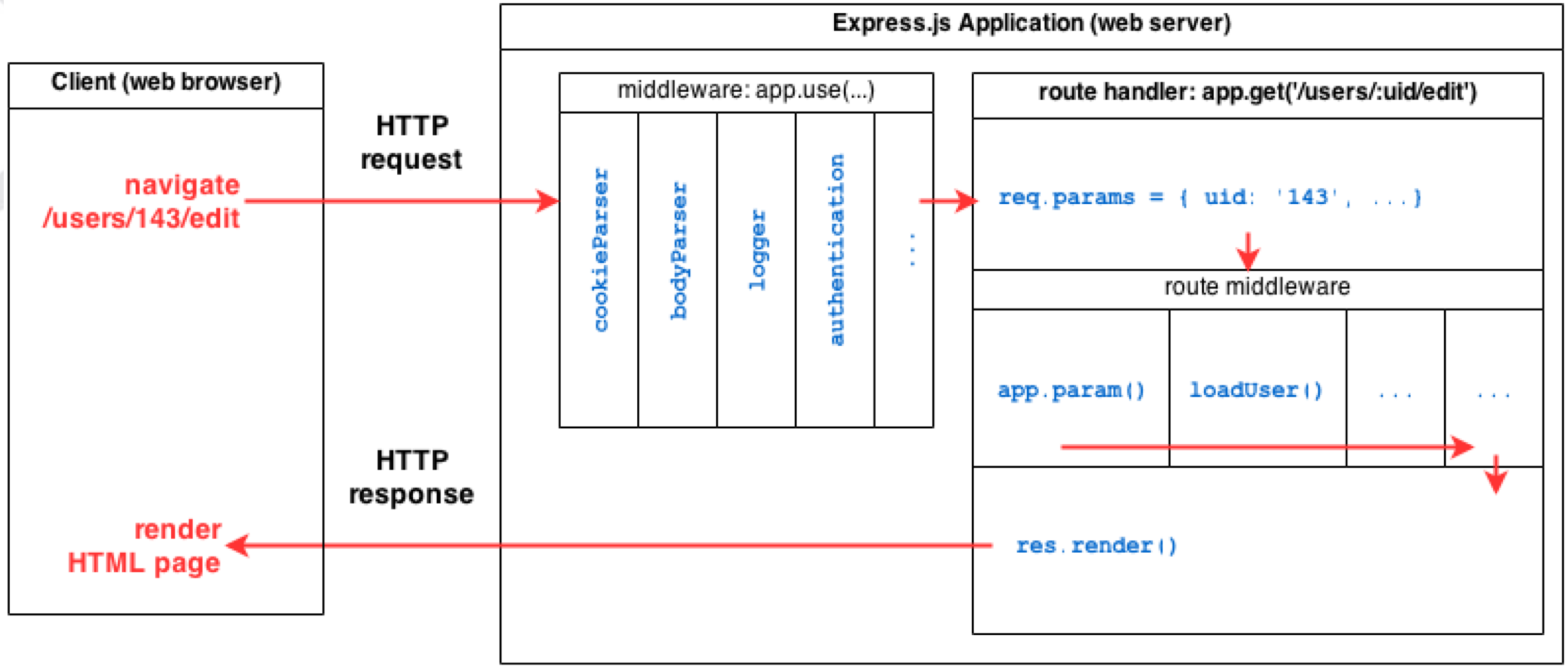
```
Route path: /flights/:from-:to
Request URL: http://localhost:3000/flights/LAX-SFO
req.params: { "from": "LAX", "to": "SFO" }
```

```
Route path: /plantae/:genus.:species
Request URL: http://localhost:3000/plantae/Prunus.persica
req.params: { "genus": "Prunus", "species": "persica" }
```

10

Middleware

ExpressJS


Express.js Application (web server)
Client (web browser)
navigate
/users/143/edit
HTTP
request
middleware: app.use(...)
cookieParser
bodyParser
logger
authentication
...
route handler: app.get('/users/:uid/edit')
req.params = { uid: '143', ...}
route middleware
app.param()
loadUser()
...
...
res.render()
HTTP
response
render
HTML page

Middleware

```
function myMiddleware(req, res, next) {
    console.log('Hello from my middleware');
    res.locals.message = "My middleware";
    next();
}

app.use(myMiddleware);

app.get('/', function(req, res) {
    res.send("Hello from Express JS & " + res.locals.message);
});
```

11

Express Router

Router

```
// index.js
app.get('/', function(req, res){…});

app.get('/products', function(req, res){…});
app.get('/products/:id', function(req, res){…});

app.get('/cart', function(req, res){…});
app.get('/cart/checkout', function(req, res){…});
…
```

JS index.js JS products.js ●

```
1   var express = require('express');
2   var router = express.Router();
3
4   router.get('/', function(req, res) {
5       res.send('View product list');
6   });
7
8   router.get('/:id', function(req, res) {
9       res.send('View product ' + req.params.id);
10  });
11
12  module.exports = router;
```

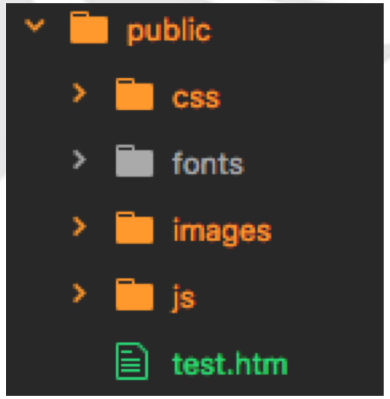
JS index.js ● JS products.js

```
1   var express = require('express');
2   var app = express();
3
4   app.get('/', function(req, res) {
5       res.send("Hello from Express JS");
6   });
7
8   var productRouter = require('./routes/products');
9   app.use('/products', productRouter);
10
11  app.listen(5000, function() {
12      console.log('Server is listening on port 5000...');
13  });
```

12

Static Files

Static Files




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

// Set Public Folder
app.use(express.static(path.join(__dirname, 'public')));
```

Request: http://IPADDRESS:3000/test.htm
Response: /public/test.htm

13

Error Handling

Error Handling

```
// Custom Error Page
// Handle 404 Not found
app.use(function(req, res) {
    res.status(404).send('Not found');
});

// Handle 500 Internal Server Error
app.use(function(error, req, res, next) {
    console.log(error);
    res.status(500).send('Internal Server Error');
});

app.listen(5000, function() {
    console.log('Server is listening on port 5000...');
});
```

14

Template Engines

Express Handlebars

- Logic-less Template engine
- Support Data Binding
- Install:

```
npm install --save express-handlebars
```

Using Express-Handlebars

```
| /views/
|--- /layouts/
|----- main.hbs
|--- /partials/
|----- header.hbs
|----- footer.hbs
|----- ... etc.
|--- index.hbs
| server.js
```

```
...
var express     = require( 'express'),
    hbs         = require( 'express-handlebars' ),
    app         = express();
...

app.engine( 'hbs', hbs( {
  extname: 'hbs',
  defaultLayout: 'main',
  layoutsDir: __dirname + '/views/layouts/',
  partialsDir: __dirname + '/views/partials/'
} ) );

app.set( 'view engine', 'hbs' );

...
```

Layouts

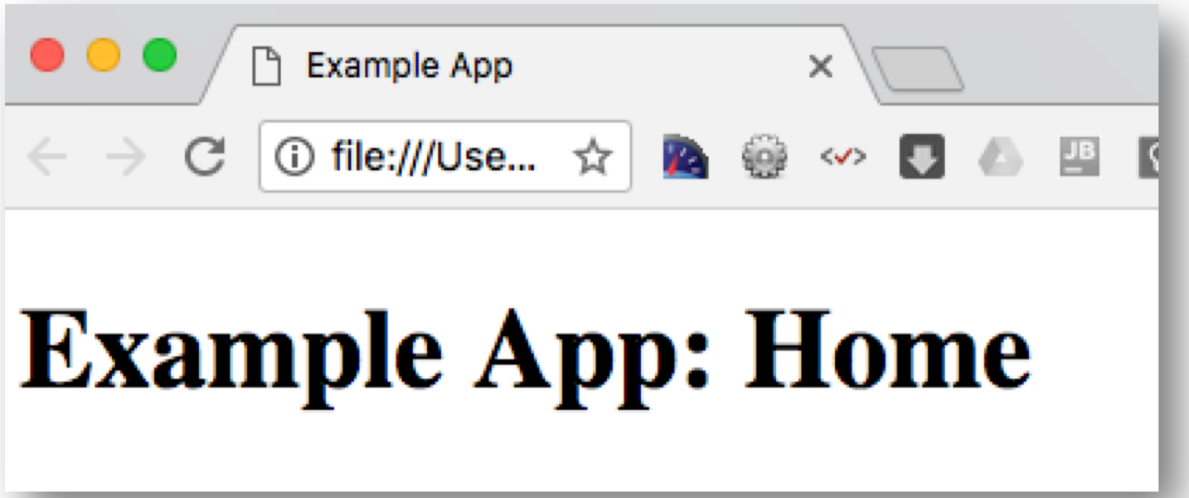
views/layouts/main.hbs

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Example App</title>
</head>
<body>

    {{{body}}}

</body>
</html>
```

views/index.hbs

```
<h1>Example App: Home</h1>
```

res.render('index');



Partials

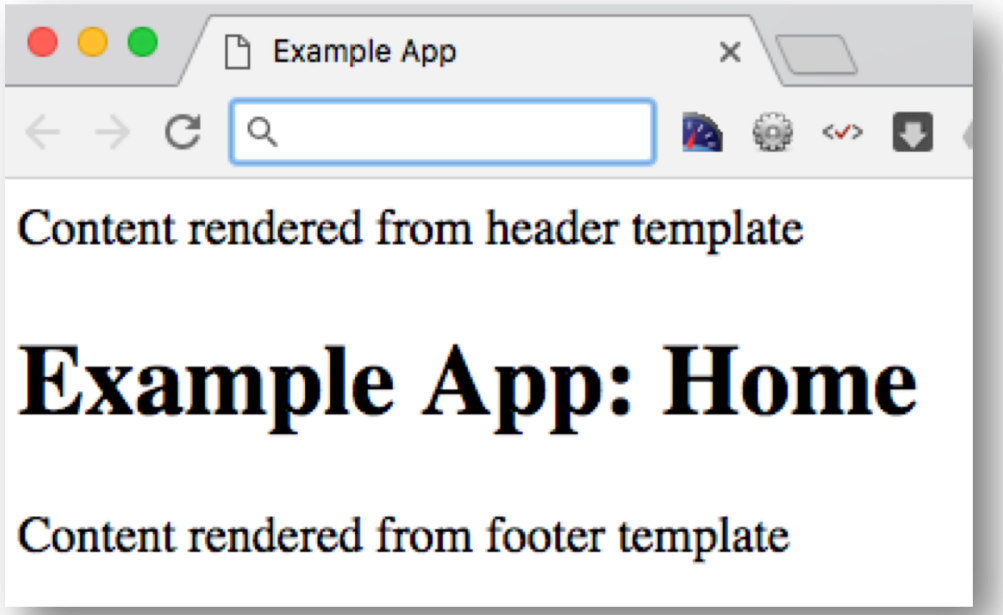
views/layouts/main.hbs

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Example App</title>
</head>
    {{> header}}
<body>

    {{> footer}}
    {{{body}}}


</body>
</html>
```

views/index.hbs

```
<h1>Example App: Home</h1>
```

res.render('index');



15

Handlebars

Handlebars

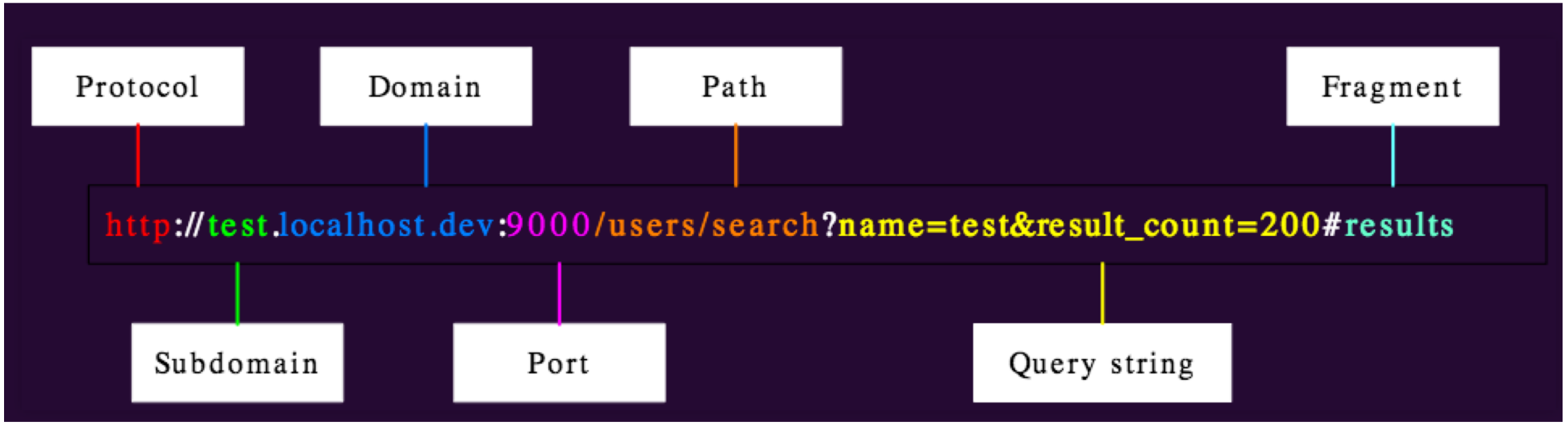
1. HTML Escape
2. Condition
3. Iteration
4. With
5. Helper
6. Comment

- `{{expression}}` vs. `{{{expression}}}`
- `{{#if}} ... {{else}} ... {{/if}}`
- `{{#each expression}} ... {{/each}}`
- `{{#with expression}} ... {{/with}}`
- `{{helper expression}}`
- `{{! Comment }}`

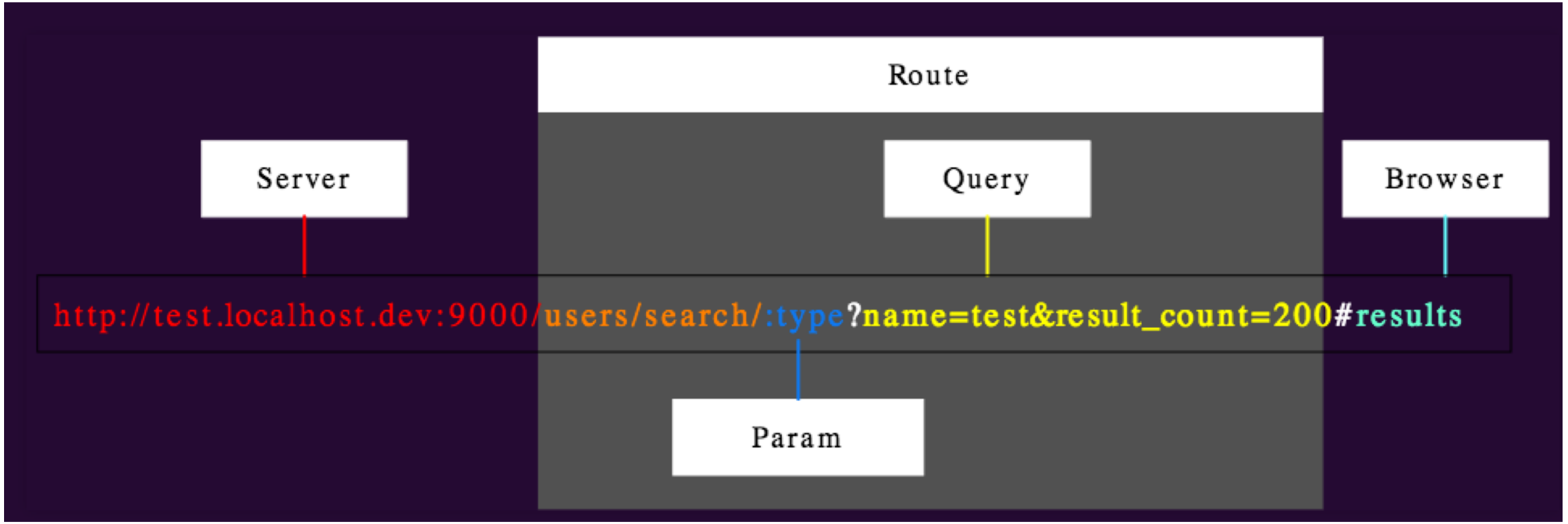
16

Query Strings

URL


Protocol
Domain
Path
Fragment
http://test.localhost.dev:9000/users/search?name=test&result_count=200#results
Subdomain
Port
Query string

URL


Route
Server
Query
Browser
http://test.localhost.dev:9000/users/search/:type?name=test&result_count=200#results
Param

17

POST Request

Route + Body

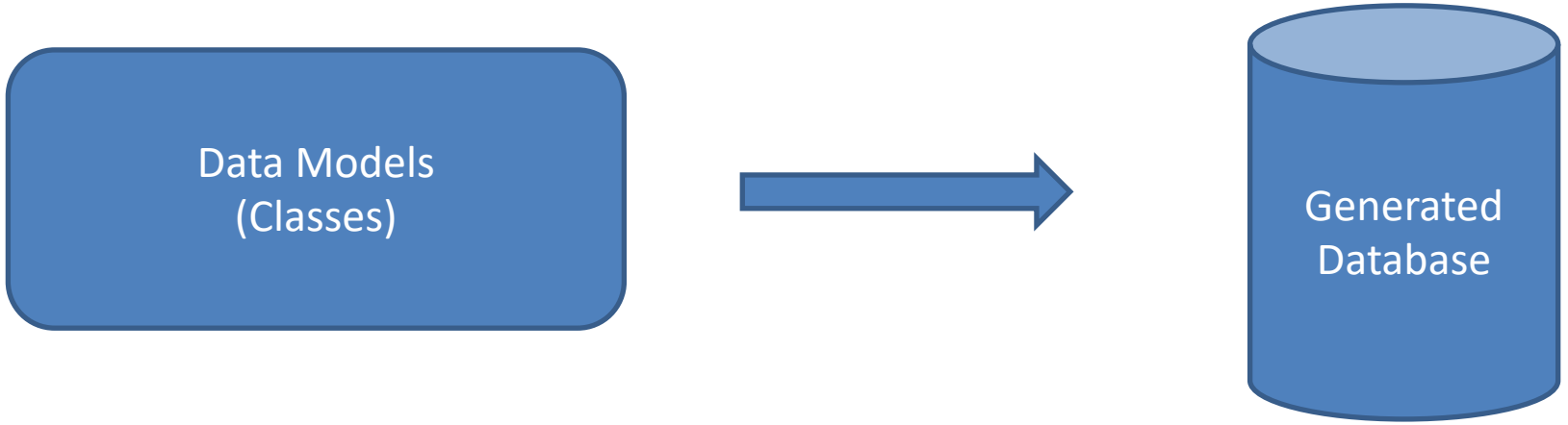
```
// html
<form action="/hello" method="post">
  <input type="text" name='location'>
  <input type="submit" value='submit'>
</form>
```

```
// Body Parser Middleware
var bodyParser = require('body-parser');
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

app.post('/hello', function(req, res) {
  res.locals.location = req.body.location;
  res.render('hello');
});
```

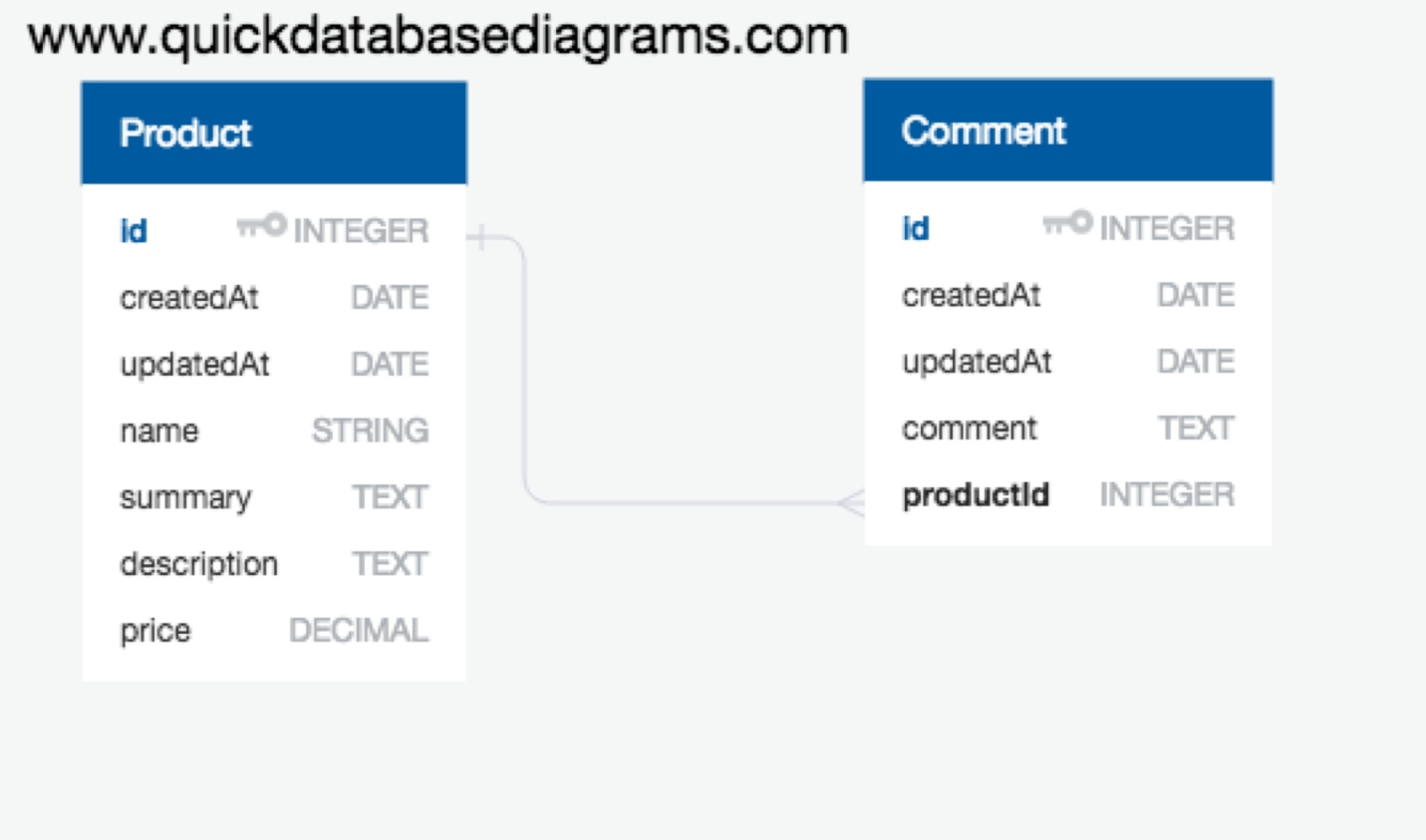
18

Sequelize-cli

Code First



Database Diagram



Sequelize

- Mapping database tables to object models
- Manipulate database: query, insert, delete, update…
- Work with different databases: PostgresSQL, MySQL, SQLite, MSSQL

```
$ npm install --save sequelize sequelize-cli

$ npm install --save pg pg-hstore
```

Sequelize-cli

- Install sequelize-cli
  - `$ npm install -g sequelize-cli`
- Init
  - `$ sequelize init`
- Generate object model
  - `$ sequelize model:create --name TableName --attributes AttributeName1:DataType,AttributeName2:DataType…`

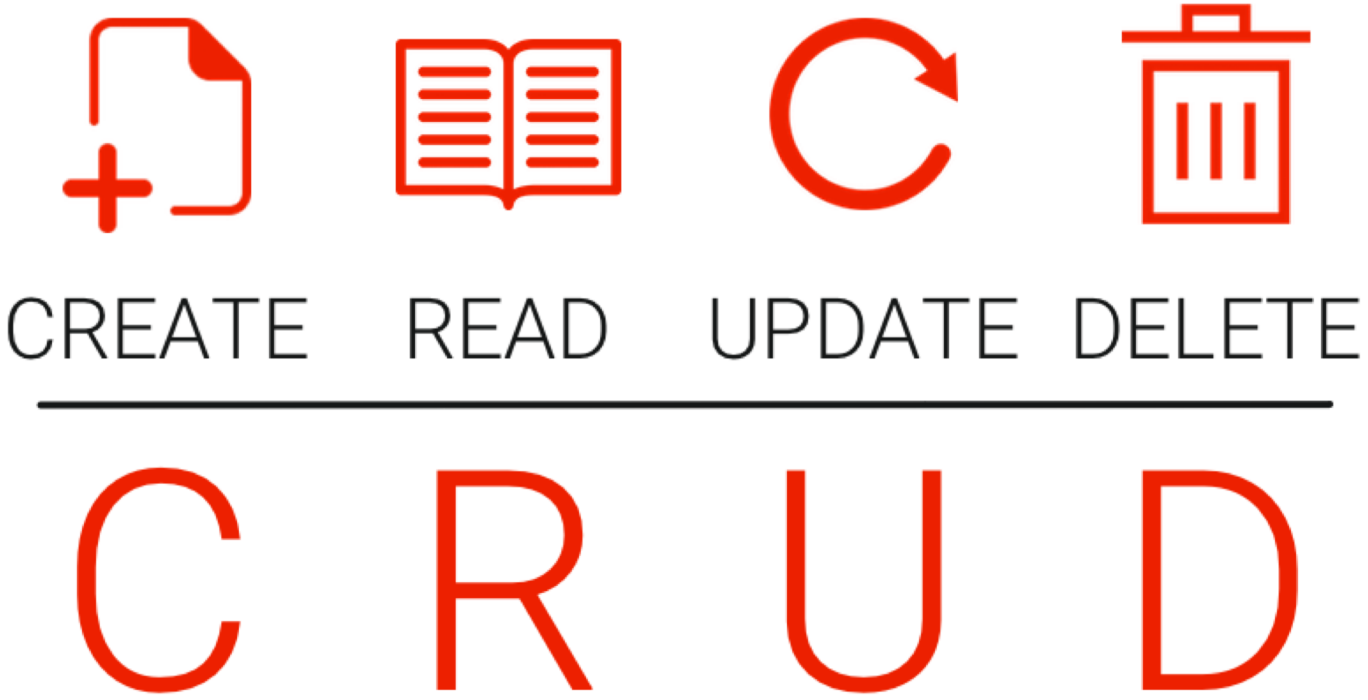
Sequelize-cli

- Generate seed file
  - `$ sequelize seed:generate --name TableName`
- Run command lines to generate seed data
  - `$ sequelize db:seed:all`
  - `$ sequelize db:seed:undo:all`
- Migration
  - `$ sequelize db:migrate`
  - `$ sequelize db:migrate:undo`
  - `$ sequelize migration:create --name FileName`
- References
  - http://docs.sequelizejs.com/

19

Sequelize

Sequelize



Create

```
1   var models = require('../models/index');
```

Then add a route for creating a new user:

```
1   router.post('/users', function(req, res) {
2     models.User.create({
3       email: req.body.email
4     }).then(function(user) {
5       res.json(user);
6     });
7   });
```

Bulk Create

```
user.bulkCreate([USERS],{ validate: true }).then(function() {
  // Congratulate user!
}).catch(function(errors) {
  // Catch  if validation failed
  // Print errors
});
```

Init – Create data

```
app.get('/init', function(req, res){
  var articles = [
    {
      title: "Design Research",
      imagepath: "/images/design.jpg",
      summary: "We help you better understand the needs
      description: "Lorem ipsum dolor sit amet, consecte
    }
  ];
  models.Article.bulkCreate(articles).then(function(){
    res.redirect('/');
  });
});
```

Find All

```
user.findAll({
   limit: 100,
   include: [{
      model: group
   }]
}).then(function(users) {
   // Send users to view
});
```

Find One

```
user.findOne({where : {email: 'johndoe@example.com'}}).then(function(user) {
  // Send user to view
});
```

Query

```
var query = {};
query.where = {
  $or: [
    {state: "California"},
    {state: "Arizona"}
  ],
  age: {
    $between: [20, 40]
  },
  lastName: {
    $ilike: '%user%'
  }
};
user.findAll(query).then(function(users) {
  // Do something awesome here
});
```

So the equivalent SQL query is

```
SELECT "id", "firstName", "lastName", "email", "age", "country", "state", "dateJoined", "crea
FROM "user" AS "user"
WHERE ("user"."state" = 'California' OR "user"."state" = 'Arizona')
  AND "user"."age" BETWEEN 20 AND 40
  AND "user"."lastName" ILIKE '%user%';
```

Operators

```
$and: {a: 5}              // AND (a = 5)
$or: [{a: 5}, {a: 6}]     // (a = 5 OR a = 6)
$gt: 6,                   // > 6
$gte: 6,                  // >= 6
$lt: 10,                  // < 10
$lte: 10,                 // <= 10
$ne: 20,                  // != 20
$eq: 3,                   // = 3
$not: true,               // IS NOT TRUE
$between: [6, 10],        // BETWEEN 6 AND 10
$notBetween: [11, 15],    // NOT BETWEEN 11 AND 15
$in: [1, 2],              // IN [1, 2]
$notIn: [1, 2],           // NOT IN [1, 2]
$like: '%hat',            // LIKE '%hat'
$notLike: '%hat'          // NOT LIKE '%hat'
$iLike: '%hat'            // ILIKE '%hat' (case insensitive) (PG only)
$notILike: '%hat'         // NOT ILIKE '%hat'  (PG only)
$like: { $any: ['cat', 'hat']}
                          // LIKE ANY ARRAY['cat', 'hat'] - also works for iLike and notLike
```

Operators

```
$and: {a: 5}              // AND (a = 5)
$or: [{a: 5}, {a: 6}]     // (a = 5 OR a = 6)
$gt: 6,                   // > 6
$gte: 6,                  // >= 6
$lt: 10,                  // < 10
$lte: 10,                 // <= 10
$ne: 20,                  // != 20
$eq: 3,                   // = 3
$not: true,               // IS NOT TRUE
$between: [6, 10],        // BETWEEN 6 AND 10
$notBetween: [11, 15],    // NOT BETWEEN 11 AND 15
$in: [1, 2],              // IN [1, 2]
$notIn: [1, 2],           // NOT IN [1, 2]
$like: '%hat',            // LIKE '%hat'
$notLike: '%hat'          // NOT LIKE '%hat'
$iLike: '%hat'            // ILIKE '%hat' (case insensitive) (PG only)
$notILike: '%hat'         // NOT ILIKE '%hat'  (PG only)
$like: { $any: ['cat', 'hat']}
                          // LIKE ANY ARRAY['cat', 'hat'] - also works for iLike and notLike
$overlap: [1, 2]          // && [1, 2] (PG array overlap operator)
$contains: [1, 2]         // @> [1, 2] (PG array contains operator)
$contained: [1, 2]        // <@ [1, 2] (PG array contained by operator)
$any: [2,3]               // ANY ARRAY[2, 3]::INTEGER (PG only)
```

Ranges

```
$contains: 2            // @> '2'::integer (PG range contains element operator)
$contains: [1, 2]       // @> [1, 2) (PG range contains range operator)
$contained: [1, 2]      // <@ [1, 2) (PG range is contained by operator)
$overlap: [1, 2]        // && [1, 2) (PG range overlap (have points in common) operator)
$adjacent: [1, 2]       // -|- [1, 2) (PG range is adjacent to operator)
$strictLeft: [1, 2]     // << [1, 2) (PG range strictly left of operator)
$strictRight: [1, 2]    // >> [1, 2) (PG range strictly right of operator)
$noExtendRight: [1, 2] // &< [1, 2) (PG range does not extend to the right of operator)
$noExtendLeft: [1, 2]  // &> [1, 2) (PG range does not extend to the left of operator)
```

Pagination / Limitation

```
// Fetch 10 instances/rows
Project.findAll({ limit: 10 })

// Skip 8 instances/rows
Project.findAll({ offset: 8 })

// Skip 5 instances and fetch the 5 after that
Project.findAll({ offset: 5, limit: 5 })
```

Association

```
// Find all projects with a least one task where task.state === project.state
Project.findAll({
    include: [{
        model: Task,
        where: { state: Sequelize.col('project.state') }
    }]
})
```

Ordering

```
Subtask.findAll({
  order: [
    // Will escape username and validate DESC against a list of valid direction parameters
    ['title', 'DESC'],

    // Will order by max(age)
    sequelize.fn('max', sequelize.col('age')),

    // Will order by max(age) DESC
    [sequelize.fn('max', sequelize.col('age')), 'DESC'],

    // Will order by  otherfunction(`col1`, 12, 'lalala') DESC
    [sequelize.fn('otherfunction', sequelize.col('col1'), 12, 'lalala'), 'DESC'],

    // Will order an associated model's created_at using the model name as the association's name.
    [Task, 'createdAt', 'DESC'],
```

Update

```
user.update({
   firstName: "John",
   lastName: "Doe",
   address: "Nevada, US"
},{
   where: { email : "johndoe@example.com" }
})
.then(function () {

});
```

DESTROY

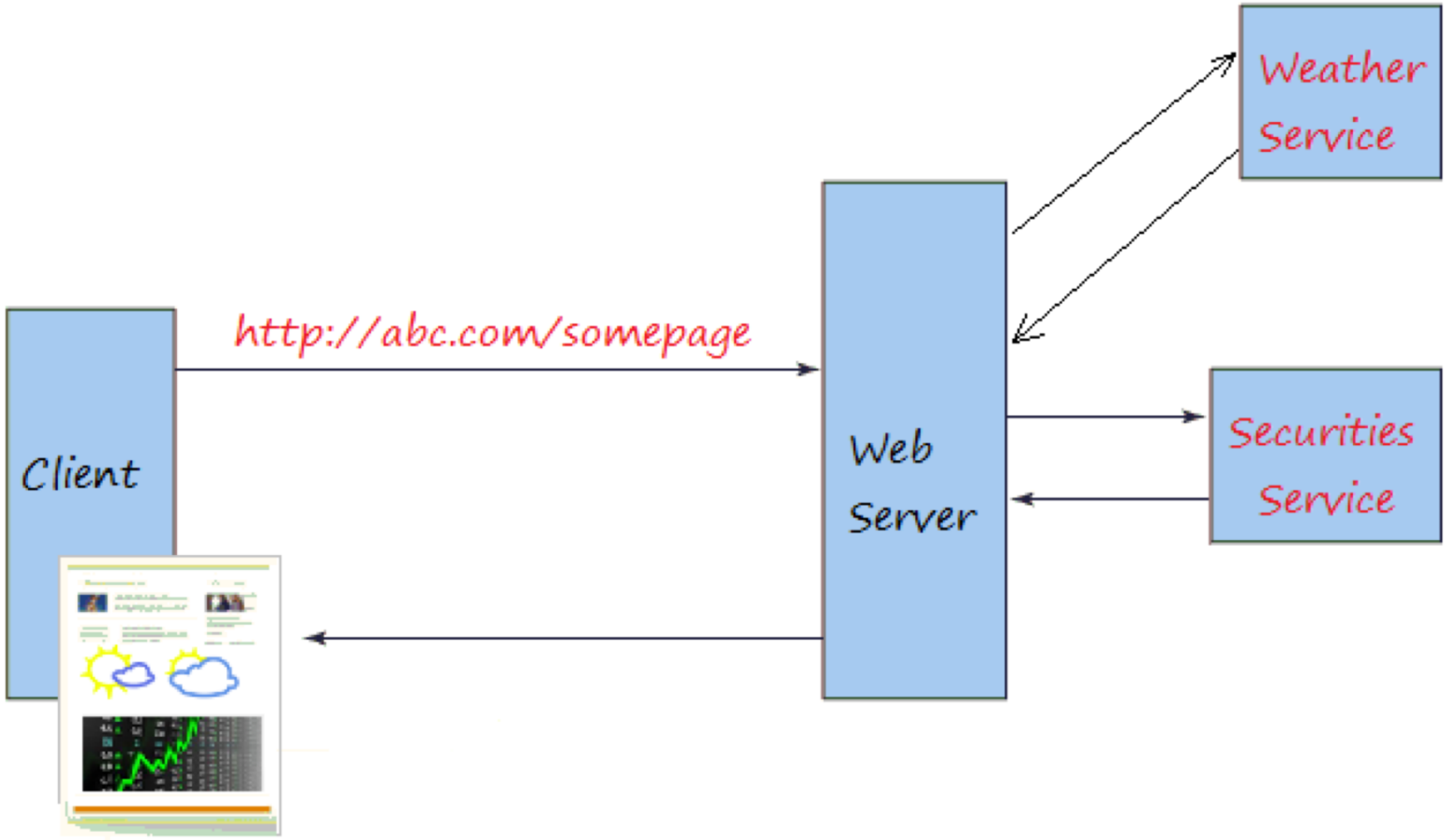
```
user.destroy({
  where: {
    email: 'johndoe@example.com'
  }
});
// DELETE FROM user WHERE email = 'johndoe@example.com';
```

20

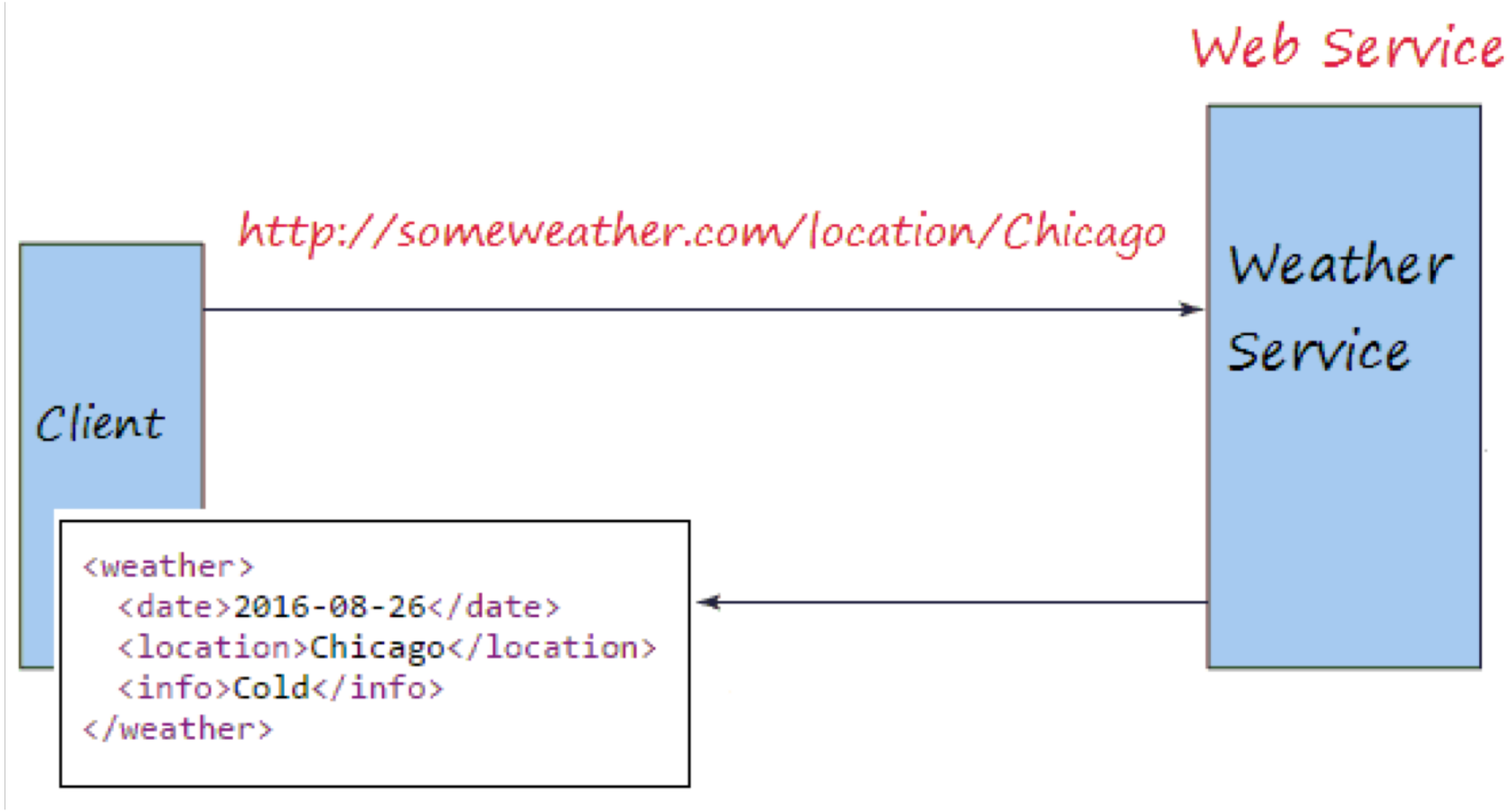
REST API

Website

Web Service

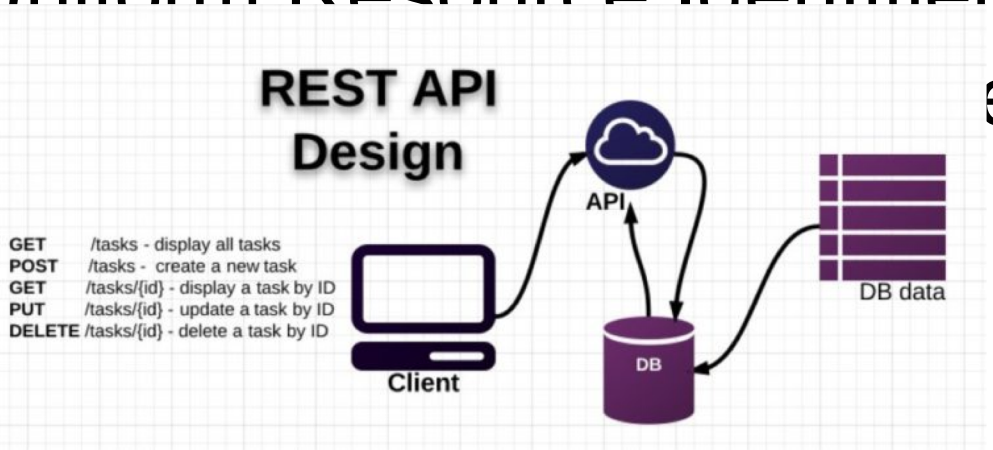

Weather Service
Client
http://abc.com/somepage
Web Server
Securities Service

Web Service


Web Service
http://someweather.com/location/Chicago
Client
Weather Service
<weather>
<date>2016-08-26</date>
<location>Chicago</location>
<info>Cold</info>
</weather>

REpresentational State Transfer

- A software architectural style that defines a set of constraints to be used for creating Web services
- Provides an Application Programming Interface between applications
- Leverages the capabilities of Hypertext Transfer Protocol (HTTP) and Uniform Resource Identifiers (URIs) to retrieve or mc e



REST API Design

- REST URI represent resource (nouns), not actions or verbs
- Use HTTP methods for actions or verbs
- REST API is Stateless
- REST API should response different types of data: json, xml, …
- Use HTTP response status codes to let client know the feedback

API Endpoints

- /getEmployee
- /getAllEmployees
- /addNewEmployee
- /updateEmployee
- /deleteEmployee
- /deleteAllEmployees

Problems:

<= contain many redundant actions

<= Burdensome to maintain when API count increases

<= URI should only contain resources (nouns) not actions or verbs

Use HTTP methods

STT	URI	HTTP Method	Post Body	Description
1	/employees	GET		Get the list of all Employees
2	/employees/john	GET		Get Employee john
3	/employees	POST	JSON String	Create one or more Employees
4	/employees/john	PUT	JSON String	Update or Create Employee john if doesn't exist
5	/employees/john	DELETE		Delete Employee john
6	/employees	DELETE		Delete Employees

RESTful API

GET PUT POST DELETE

REST URI contain resources (nouns) not actions or verbs

- The resource should always be **plural** in API endpoint, resource can be under other resources
  - GET /companies/3/employees
  - POST /companies/3/employees
  - PUT /companies/3/employees/john
  - DELETE /companies/3/employees/john
- DO NOT use ACTIONS or VERBS in API Endpoint
  - ~~GET /getAllEmployees~~
  - ~~POST /addNewEmployee~~
  - ~~DELETE /deleteEmployee~~

Using query strings for filtering, Sorting, Searching, and Pagination

- Sorting
  - GET /companie?sort=rank_asc
- Filtering
  - GET /companies?category=banking&location=asia
- Searching
  - GET /companies?search=Digital Mckinsey
- Pagination
  - GET /companies?page=23

HTTP Response Status Code

- 2xx (Success)
  - 200 OK
  - 201 Created
  - 204 No Content (vd Delete)
- 5xx (Server Error)
  - 500 Internal Server Error
  - 503 Service Unavailable
- 4xx (Client Error)
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
  - 410 Gone

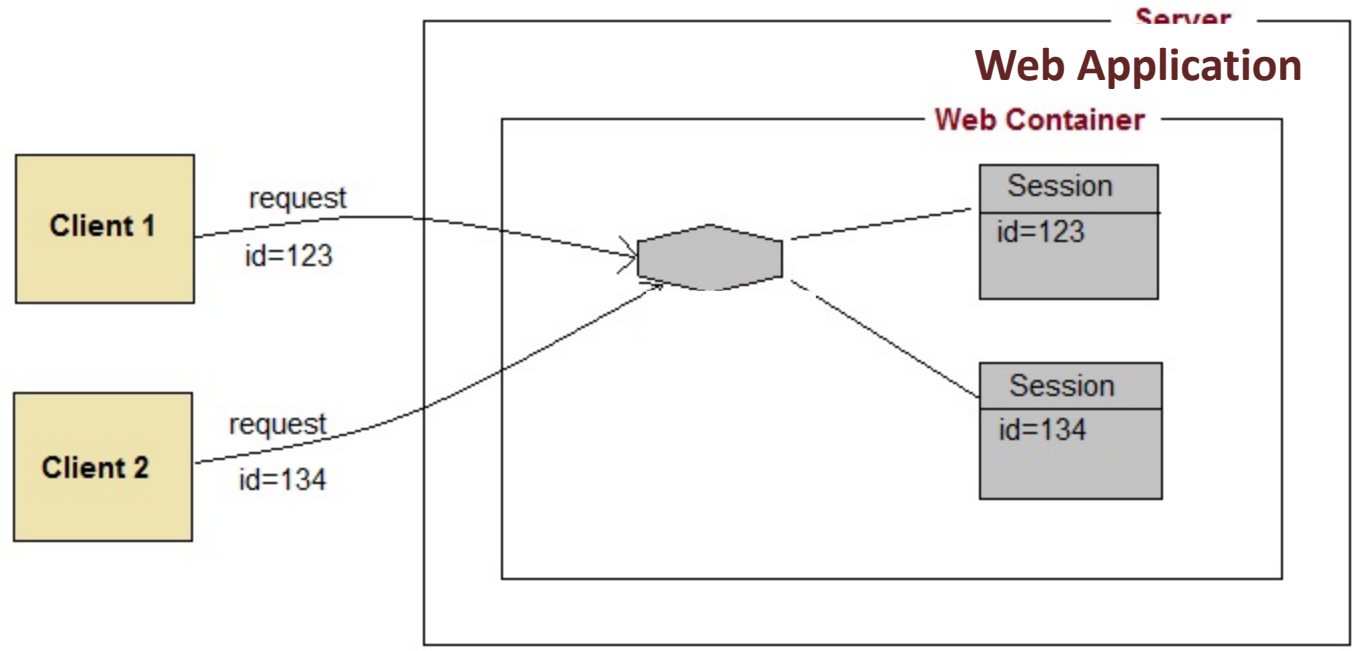
21

Session

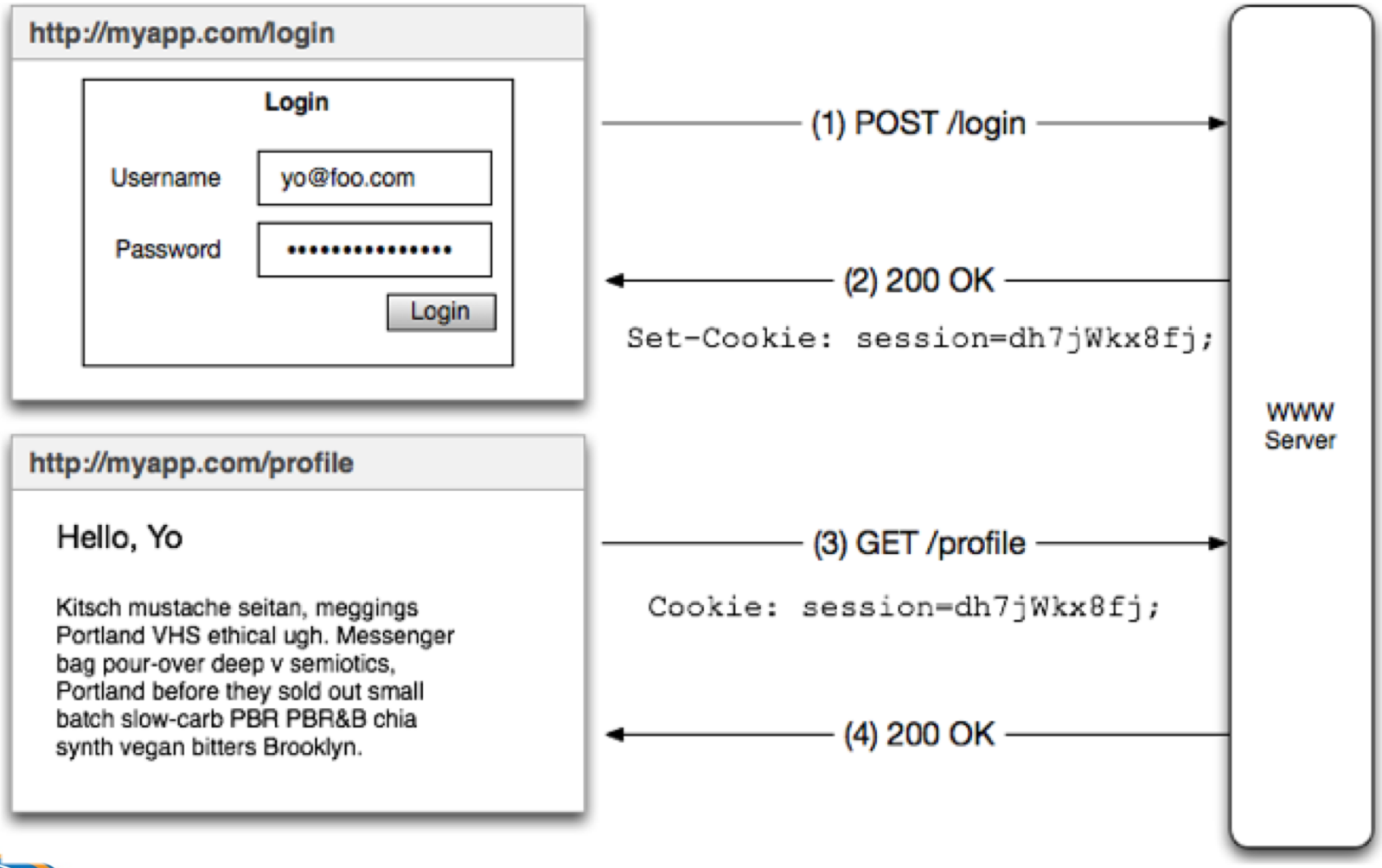
Session

- **A server-side storage** of information that is desired to **persist throughout the user's interation** with the web application
- **Session uses cookie to store a unique identifier** for each client (called a "session id")
- Session id is passed to the web server every time the browser make an HTTP request

Session


Server
Web Application
Web Container
Client 1
request
id=123
Client 2
request
id=134
Session
id=123
Session
id=134

Example


http://myapp.com/login
Login
Username
yo@foo.com
Password
Login
(1) POST /login
(2) 200 OK
Set-Cookie: session=dh7jWkx8fj;
http://myapp.com/profile
Hello, Yo
Kitsch mustache seitan, meggings Portland VHS ethical ugh. Messenger bag pour-over deep v semiotics, Portland before they sold out small batch slow-carb PBR PBR&B chia synth vegan bitters Brooklyn.
(3) GET /profile
Cookie: session=dh7jWkx8fj;
(4) 200 OK
WWW Server

Using Session

```
var express = require('express');
var sessions = require('express-session');
var app = express();

app.use(sessions({
  cookie: { httpOnly: true, maxAge: null},
  secret: 'your secret',
  resave: false,
  saveUninitialized: false
}));
```

Session Options

- cookie
  - Setting object for the session ID cookie
- secret (required option)
  - The secret used to sign the session ID cookie
- resave [true]
  - Forces the session to be saved back to the session store
- saveUninitialized [true]
  - Forces a session that is “uninitialized” to be saved to the store

Save Session Information

```
app.post('/login', function(req, res){
  var username = req.body.username;
  var password = req.body.password;
  if (username == "admin" && password == "admin") {
    req.session.username = username;
    res.redirect('/admin');
  } else {
    res.render('pages/login');
  }
});
```

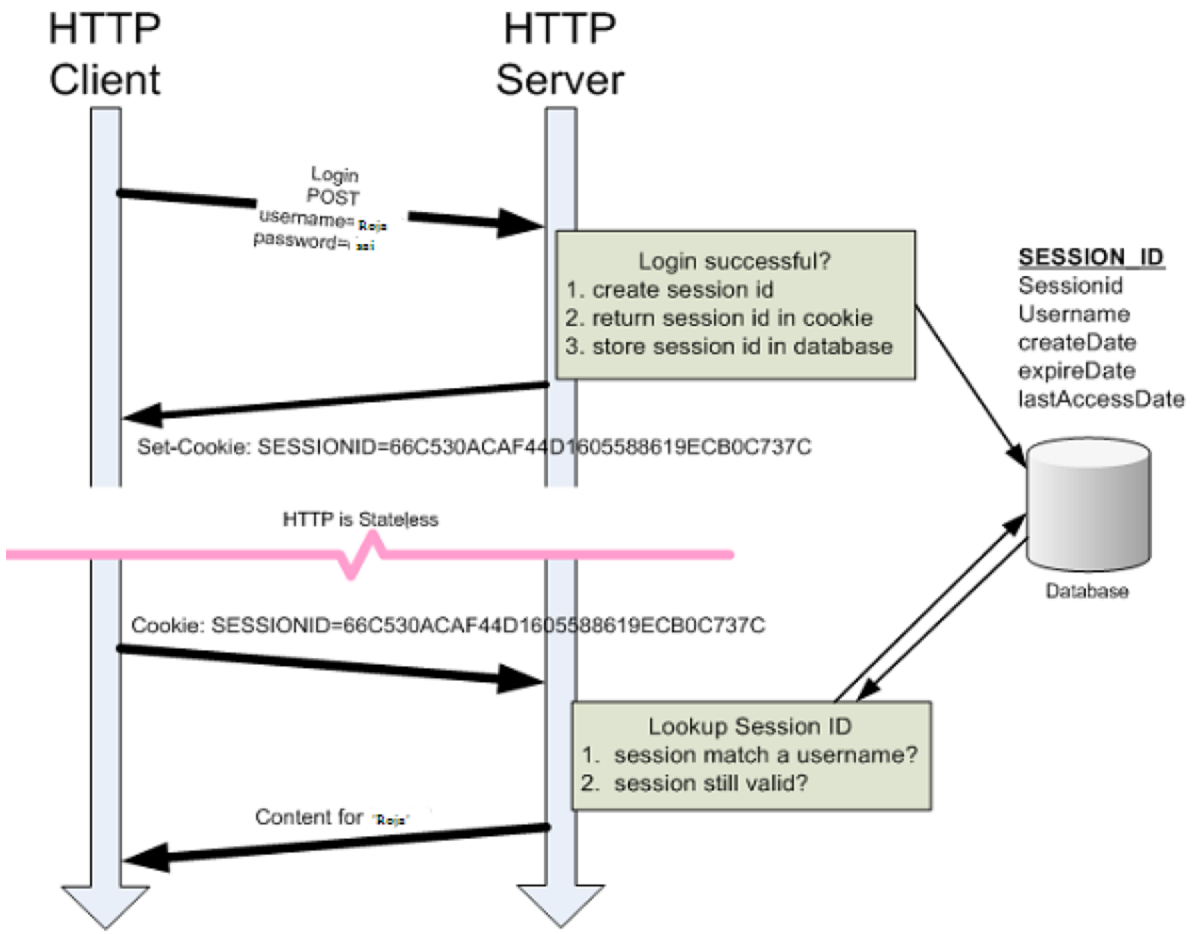
Read Session Information

```
app.get('/admin', function(req, res){
  if (req.session.username) {
    res.render('pages/admin');
  } else {
    res.redirect('/login');
  }
});
```

Destroy Session Information

```
app.get('/logout', function(req, res){
  req.session.destroy(function(error){
      if (error) return next(error);
      res.redirect('/login');
  });
```

Session Store



connect-pg-simple

```
var pg = require('pg')
  , session = require('express-session')
  , pgSession = require('connect-pg-simple')(session);

app.use(session({
  store: new pgSession({
    pg : pg,                                  // Use global pg-module
    conString : process.env.FOO_DATABASE_URL, // Connect using something else than default DATABASE_URL
    tableName : 'user_sessions'               // Use another table-name than the default "session" one
  }),
  secret: process.env.FOO_COOKIE_SECRET,
  resave: false,
  cookie: { maxAge: 30 * 24 * 60 * 60 * 1000 } // 30 days
}));
```

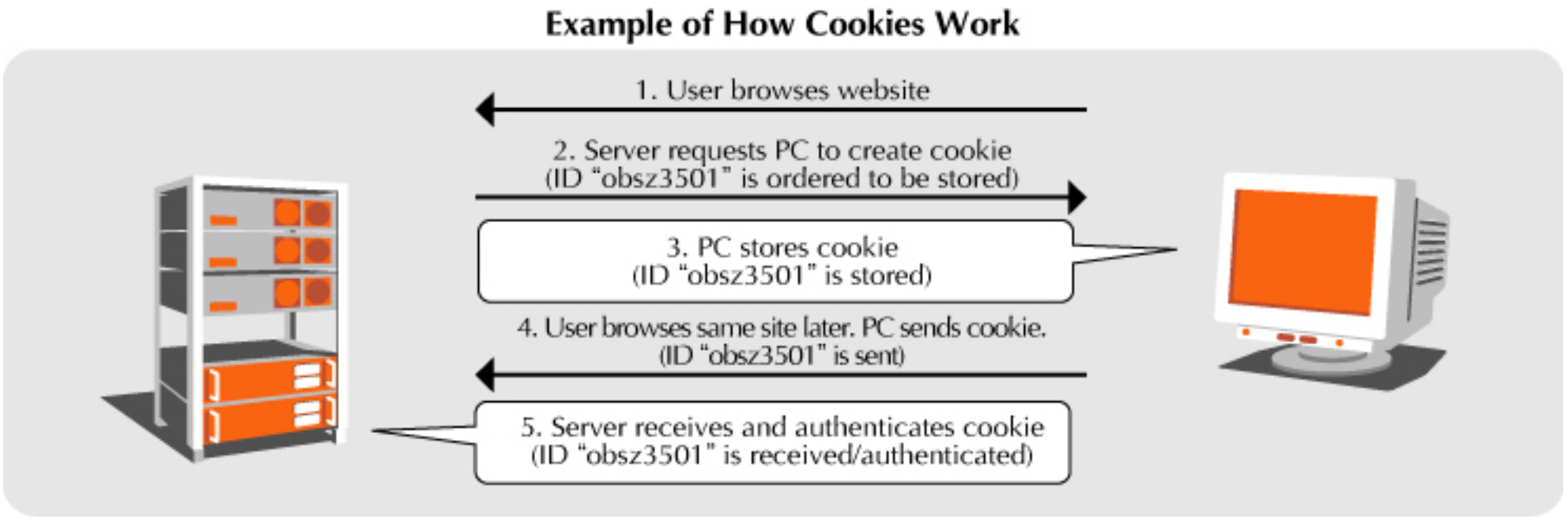
22

Cookies

Cookies

- A small piece of text **stored on a user's computer** by their browser
- Is passed to the web server every time the browser make an HTTP request
- Common uses:
  - Authentication
  - Storing of site preferences
  - Shopping cart items
  - Server session identification

How Cookies Work



Note: Don't exceed **50 cookies or and 4093 bytes per domain**

Check Browser Cookie Limits:

http://browsercookielimits.squawky.net/

Using Cookies in Node.JS

```
var express = require('express');
var cookieParser = require('cookie-parser');

var app = express();
app.use(cookieParser());
```

Set & Get Cookies in Node.JS

```
app.post('/login', function (req, res){
  var username = req.body.username;
  res.cookie('username' , username,
          { maxAge: 60 * 60 * 24 * 7, /* 1 week */
            httpOnly: true
          });
  ...
});
```

```
app.get('/login', function(req, res){
  var username = req.cookies.username || "";
  ...
});
```

Clear Cookie

```
app.get('/clearCookies', function(req, res, next){
  res.clearCookie('username');
  res.send('Clear Cookies!')
});
```

Cookie Options

- domain & path [‘/”]
  - Define the scope of the cookie
- maxAge [null]
  - Specifies the number (in seconds) to use when calculating expired date
- httpOnly [true]
  - By default set true to tell browsers to not allow client side script access to the cookie (if the browser supports it)
- Secure [false]
  - Set true if use HTTPS

Using JSON to store Object in Cookie

Set Cookie:

res.cookie('cart', JSON.stringify({ styles : styles[product], count : 0, total : 0 }))

Get Cookie

```
var cookieData = JSON.parse(req.cookies.cart)
```