



# NODE.JS

Part II

# NEED FOR A TEMPLATE

- ☐ If you want to add specific handling for different HTTP verbs (e.g. GET, POST, DELETE, etc.)
- ☐ Separately handle requests at different URL paths ("routes")
- ☐ Serve static files, or use templates to dynamically create the response,
- ☐ Node won't be of much use on its own.
- ☐ You will either need to write the code yourself, or you can avoid reinventing the wheel and use a web framework!

# APPLICATION FRAMEWORKS

- ❑ provides frozen spots
  - ❑ overall architecture
  - ❑ How the components interact
- ❑ allows to concentrate in hot spots to extend the behaviour of the framework
  - ❑ Hot spots are the functions written for the application
- ❑ A framework is not suitable for a problem when ...

# WEB APPLICATION FRAMEWORKS

- ❑ An application framework that is designed to support development of web applications that generally includes
  - ❑ Database support
  - ❑ Templating framework for generating dynamic web content
  - ❑ HTTP session management with middleware support
  - ❑ Built-in testing framework
- ❑ It can also support internationalization, security and privacy
- ❑ Consistent look and feel and consistent with database

# WEB FRAMEWORKS EXAMPLES

- ❑ Ruby on Rails
- ❑ Play
- ❑ ASP.NET
- ❑ Django
- ❑ Symfony
- ❑ Spring
- ❑ Vue.js
- ❑ Angular js

# EXPRESS

- ❑ Express is the most popular *Node* web framework, and is the underlying library for a number of other popular Node web frameworks.
- ❑ It provides mechanisms to:
- ❑ Write handlers for requests with different HTTP verbs at different URL paths (routes).
- ❑ Integrate with "view" rendering engines in order to generate responses by inserting data into templates.
- ❑ Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
- ❑ Add additional request processing "middleware" at any point within the request handling pipeline.

# EXPRESS FEATURES

- ❑ *Express* itself is fairly minimalist
- ❑ Developers have created compatible middleware packages to address almost any web development problem
- ❑ There are libraries to work with cookies, sessions, user logins, URL parameters, POST data, security headers, and *many* more.
- ❑ You can find a list of middleware packages maintained by the Express team at [Express Middleware](#) (along with a list of some popular 3rd party packages).

# IS EXPRESS OPINIONATED

- ❑ Opinionated frameworks are those with opinions about the "right way" to handle any particular task. They often support rapid development *in a particular domain* (solving problems of a particular type) because the right way to do anything is usually well-understood and well-documented
- ❑ Unopinionated frameworks, by contrast, have far fewer restrictions on the best way to glue components together to achieve a goal, or even what components should be used.



# EXPRESS OPINION

- ❑ Express is unopinionated.
- ❑ You can insert almost any compatible middleware you like into the request handling chain, in almost any order you like.
- ❑ You can structure the app in one file or multiple files, using any directory structure.
- ❑ You may sometimes feel that you have too many choices

# EXPRESS FEATURES

- ☐ Express provides methods to specify what function is called for a particular HTTP verb (GET, POST, SET, etc.) and URL pattern ("Route")
- ☐ Provides methods to specify what template ("view") engine is used
- ☐ Where template files are located
- ☐ What template to use to render a response
- ☐ You can use Express middleware to add support for cookies, sessions, and users, getting POST/GET parameters, etc.
- ☐ You can use any database mechanism supported by Node (Express does not define any database-related behavior).

# EXPRESS APP

- ❑ This object, which is traditionally named `app`, has methods for
  - ❑ routing HTTP requests
  - ❑ configuring middleware,
  - ❑ rendering HTML views,
  - ❑ registering a template engine, and
  - ❑ modifying application settings that control how the application behaves (e.g. the environment mode, whether route definitions are case sensitive, etc.)
- ❑ `res.json()` to send a JSON response or `res.sendFile()` to send a file

```
const express = require("express");
const app = express();
const port = 3000;

app.get("/", function (req, res) {
  res.send("Hello World!");
});

app.listen(port, function () {
  console.log(`Example app listening on port ${port}!`);
});
```

# REQUEST HANDLING

A common convention for Node and Express is to use error-first callbacks. In this convention, the first value in your *callback functions* is an error value, while subsequent arguments contain success data.

- ❑ The *Express application* object also provides methods to define route handlers for all the other HTTP verbs, which are mostly used in exactly the same way:
- ❑ `checkout()`, `copy()`, **`delete()`**, **`get()`**, `head()`, `lock()`, `merge()`, `mkactivity()`, `mkcol()`, `notify()`, `options()`, `patch()`, **`post()`**, `purge()`, **`put()`**, `report()`, `search()`, `subscribe()`, `trace()`, `unlock()`, `unsubscribe()`
- ❑ There is a special routing method, `app.all()`, which will be called in response to any HTTP method.
- ❑ This is used for loading middleware functions at a particular path for all request methods.

```
app.all("/secret", function (req, res, next) {  
    console.log("Accessing the secret section...");  
    next(); // pass control to the next handler });
```

# CREATING THE SERVER

```
const express = require('express');
const http = require('http');
const app = express();
const server = http.Server(app);
// Configuration
server.listen(process.env.PORT || 8000, () => {
  console.log(`[ server.js ] Listening on port
  ${server.address().port}`);
});
```

- **Express** allows us to create the webserver and REST API needed to run our app
  - Express is used here to create the http server and attaching it to an Express app
- **Socket.io** is the real-time engine (utilizing the WebSocket API)

## PATH TO GET REQUEST

```
1  const express = require('express');
2  const http = require('http');
3  const path = require('path');
4  const app = express();
5  const server = http.Server(app);
6
7  // Configuration
8
9  server.listen(process.env.PORT || 8000, () => {
10    console.log(`[ server.js ] Listening on port ${server.address().port}`);
11  });
12
13  // Routes
14
15  app.get('/', (req, res) => {
16    res.sendFile(path.join(__dirname, 'views/show.html'));
17  });
18
19  app.get('/edit', (req, res) => {
20    res.sendFile(path.join(__dirname, 'views/edit.html'));
21  });
```

# ROUTING THROUGH EXPRESS

```
const wiki = require("./wiki.js");  
  
app.use("/wiki", wiki);  
  
// Home page route  
router.get("/", function (req, res) {  
    res.send("Wiki home page");  
});
```

- ❑ Often it is useful to group route handlers for a particular part of a site together and access them using a common route-prefix
- ❑ a site with a Wiki might have all wiki-related routes in one file and have them accessed with a route prefix of `/wiki/`
- ❑ In *Express* this is achieved by using the [express.Router](#) object.

# MIDDLEWARE

- ❑ Middleware is used extensively in Express apps, for tasks from serving static files to error handling, to compressing HTTP responses
- ❑ Route functions end the HTTP request-response cycle by returning some response to the HTTP client, middleware functions *typically* perform some operation on the request or response and then call the next function in the "stack", which might be more middleware or a route handler
- ❑ The order in which middleware is called is up to the app developer.
- ❑ It is almost always the case that middleware is called before setting routes, or your route handlers will not have access to functionality added by your middleware.
- ❑ The **only** difference between a middleware function and a route handler callback is that middleware functions have a third argument next, which middleware functions are expected to call if they are not that which completes the request cycle



# MIDDLEWARE IN EXPRESS

- ❑ To use third party middleware you first need to install it into your app using npm

```
npm install morgan
```

- ❑ You could then call `use()` on the *Express application object* to add the middleware to the stack

```
// Function added with use() for all routes and verbs  
app.use(a_middleware_function);
```

```
// Function added with use() for a specific route  
app.use("/someroute", a_middleware_function);
```

```
// A middleware function added for a specific HTTP verb and  
route app.get("/", a_middleware_function);
```

# SERVING STATIC CONTENT

- ❑ Any files in the public directory are served by adding their filename (*relative to the base "public" directory*) to the base URL

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

- ❑ The URLs can be

```
http://localhost:3000/images/dog.jpg
```

```
http://localhost:3000/css/style.css
```

- ❑ If a file cannot be found by one middleware function then it will be passed on to the subsequent middleware (the order that middleware is called is based on your declaration order).

```
app.use(express.static("media"));
```

## ALL ABOUT ERRORS

- ❑ Errors are handled by one or more special middleware functions that have four arguments, instead of the usual three: (err, req, res, next)

```
app.use(function (err, req, res, next) {  
  console.error(err.stack);  
  res.status(500).send("Something broke!"); });
```

# TEMPLATES

- ❑ Template engines (also referred to as "view engines") allow you to specify the *structure* of an output document in a template, using placeholders for data that will be filled in when a page is generated.
- ❑ Templates are often used to create HTML, but can also create other types of documents
- ❑ At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client

# HTML TEMPLATE ENGINES

- ❑ Some popular template engines that work with Express are [Pug](#), [Mustache](#), and [EJS](#).
- ❑ The [Express application generator](#) uses [Jade](#) as its default, but it also supports several others
- ❑ In your application settings code you set the template engine to use and the location where Express should look for templates using the 'views' and 'view engine' settings

- ❑ Directory to keep the template files

```
app.set('views', './views')
```

- ❑ To set pug as the template engine

```
app.set('view engine', 'pug')
```

- ❑ The view engine cache does not cache the contents of the template's output, only the underlying template itself. The view is still re-rendered with every request even when the cache is on.

# PUG ENGINE

```
html
  head
    title= title
  body
    h1= message
    a(href = url) Homepage
```

```
app.get("/views", function
(req, res) {
  res.send("Hello World!");
});
```

```
app.get('/views', (req, res) => {
  res.render('index.pug', { title: 'Hey',
message: 'Hello there!' })
})
```

# SETTING CONDITIONALS

html

head

title Simple template

body

if(user)

h1 Hi, #{user.name}

else

a(href = "/wiki") Sign Up

```
router.get("/about", function (req, res) {  
  res.render('signup.pug', {message: 'Wiki  
home page',  
url:"http://www.tutorialspoint.com"});  
});
```

```
router.get("/about", function (req, res) {  
  res.render('signup.pug', {{user: {name:  
"Aritra", age: "22"}  
}});
```

## ROUTING THROUGH THE FOLDERS

- ❑ In pug file insert the image source

```
img(src = '/testImage.jpg' alt = 'Testing Image')
```

- ❑ Insert static content

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

- ❑ Adding a virtual path for the static content

```
app.use('/static', express.static('views'));
```




# PUSHING DATA

Manual- User may click on “refresh” when (s)he has decided to pull the data

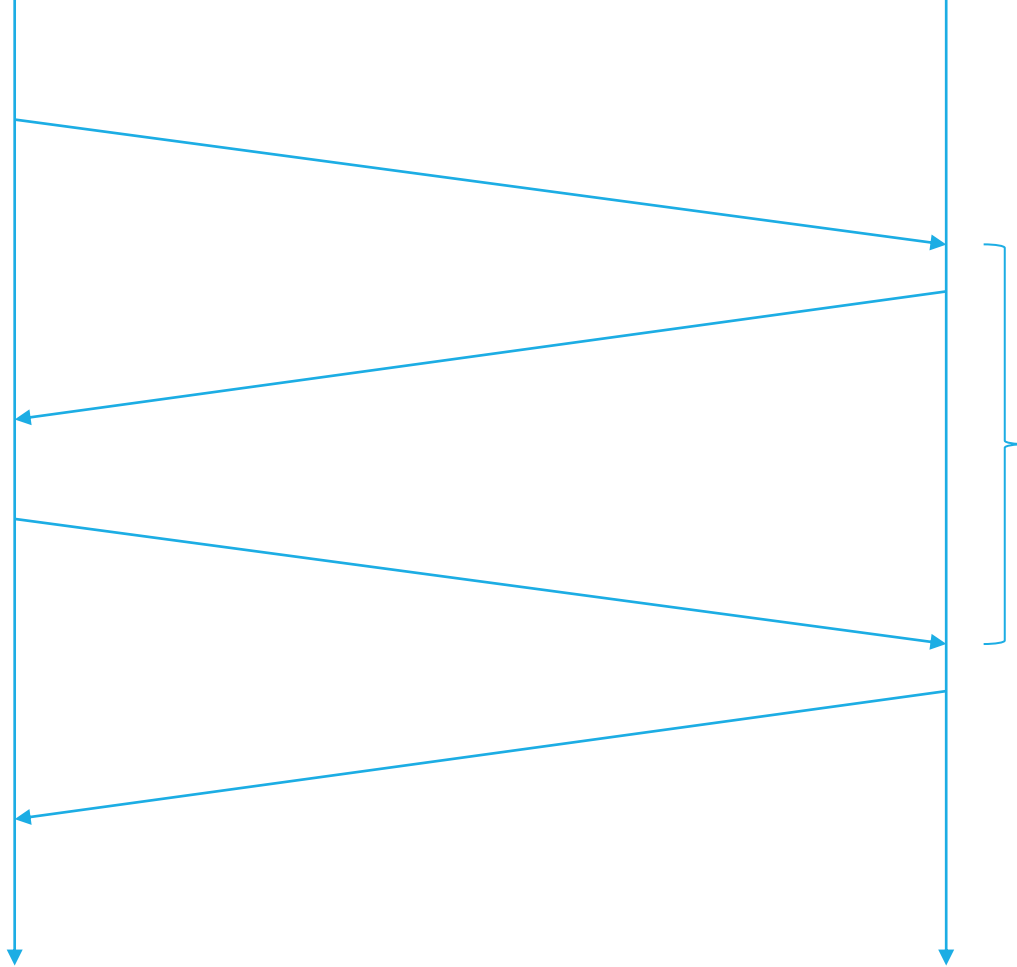
Every time the window is opened data is pulled from the server

Periodic update

- Overhead on the server processes as it needs to process the request even when no update is to be notified
  - Polling
  - Exponential backoff
- 

Client

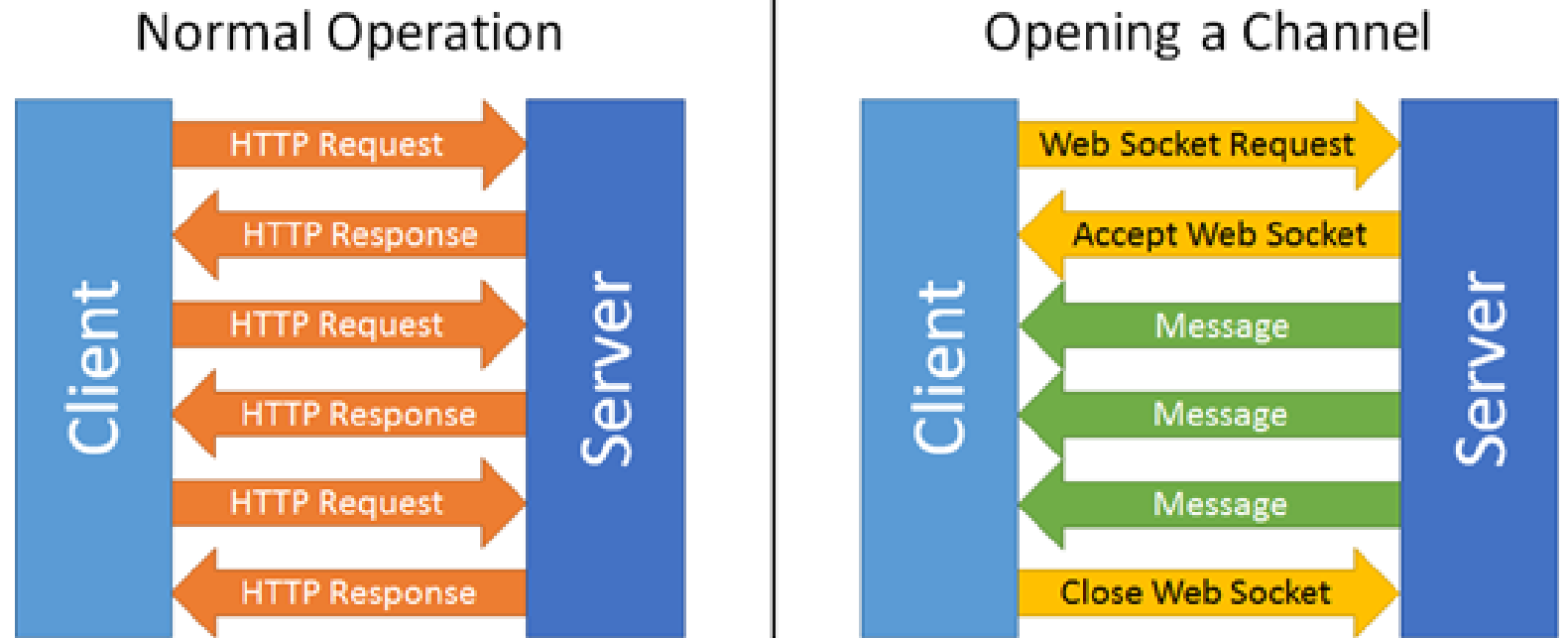
Server



PERIODIC UPDATE

<https://dzone.com/articles/thoughts-on-server-sent-events-http2-and-envoy-1>

# WEBSOCKETS



- ❑ communication protocol which features bi-directional, full-duplex communication over a persistent TCP connection
- ❑ Any party can push data anytime
- ❑ Single TCP connection for full duplex traffic
- ❑ Message transfer on websockets does not require all parts of HTTP to be sent (header, URL, content type, body etc.)
- ❑ Simply send binary messages or some other format back and forth in a server
- ❑ By default, port 80 is used
- ❑ Port 443 is used for connection tunneled over the TLS

# WEB SOCKET HTTP COMPATIBILITY



# WEB SOCKET ADVANTAGES

Stateful connection

Message overhead of polling is more than web socket

STOMP- Simple Text Oriented Messaging Protocol

<https://spring.io/guides/gs/messaging-stomp-websocket/>

S. El Mimouni and M. Bouhdadi, "Formal modeling of the Simple Text Oriented Messaging Protocol using Event-B method," 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA), 2015, pp. 1 -4, doi: 10.1109/AICCSA.2015.7507170.

# WEB SOCKET PROBLEMS

Web sockets enable a server to push data only if the client is connected

Web sockets are difficult to synchronize with more clients

Keeping an open connection can have substantial resource impact

For shared hosting servers, web socket is not a scalable option

http responses can be cached by browser or by proxies

- There is no such built-in mechanism for requests sent via webSockets