Mojolicious
and websockets

Erik Johansen
DK Hostmaster A/S
Mojoconf 2014
2014-05-24

Mojolicious
and websockets

1) Basics
    a) Websockets
2) Two examples
    a) Send message, every 3 sec.
    b) Chat script – multiple clients talking together.

1) Basics

Websockets

`ws://www.server.dk/`
`wss://www.server.dk/`

- Part of HTML5
- Full-duplex communication over single TCP connection.
- Supported by all latest browsers (except for Opera Mini).
- Idle when no data. Eliminates polling.
- Less overhead (cookies, authentication is sent only once)

Websockets

`ws://www.server.dk/`

`wss://www.server.dk/`

- Part of HTML5
- Full-duplex communication over single TCP connection.
- Supported by all latest browsers (except for Opera Mini).
- Idle when no data. Eliminates polling.
- Less overhead (cookies, authentication is sent only once)

Websockets

`ws://www.server.dk/`
`wss://www.server.dk/`

- Part of HTML5
- Full-duplex communication over single TCP connection.
- Supported by all latest browsers (except for Opera Mini).
- Idle when no data. Eliminates polling.
- Less overhead (cookies, authentication is sent only once)

Websockets

`ws://www.server.dk/`
`wss://www.server.dk/`

- Part of HTML5
- Full-duplex communication over single TCP connection.
- Supported by all latest browsers (except for Opera Mini).
- Idle when no data. Eliminates polling.
- Less overhead (cookies, authentication is sent only once)

Websockets

Websockets can be used for a variety of web applications:

- Games
- Stock tickers
- Multiuser applications with simultaneous editing
- User interfaces exposing server-side services in real time,
- Etc.

Quoted from RFC 6455

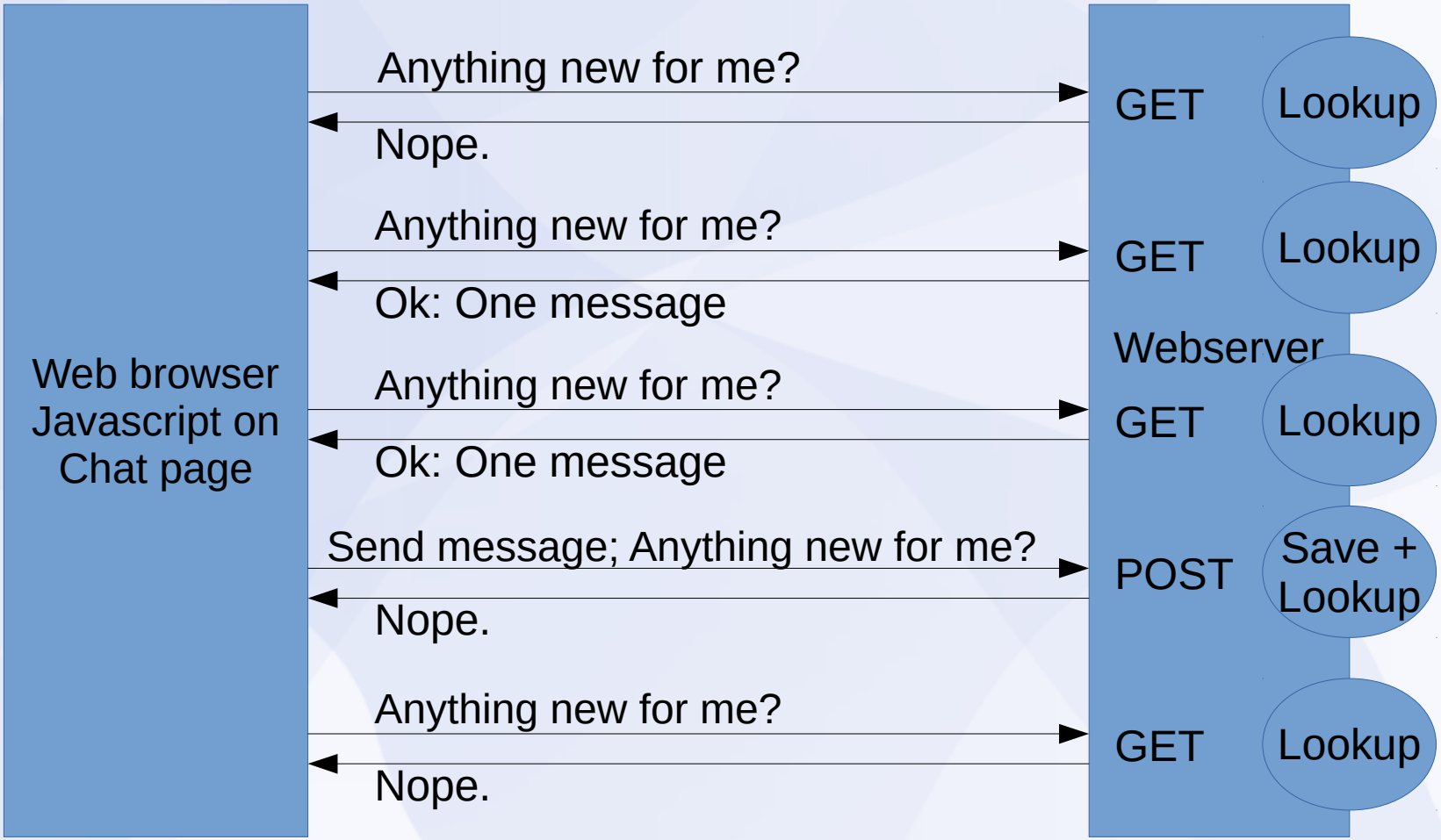
Websockets
without websockets

Polling

Long polling

Poll

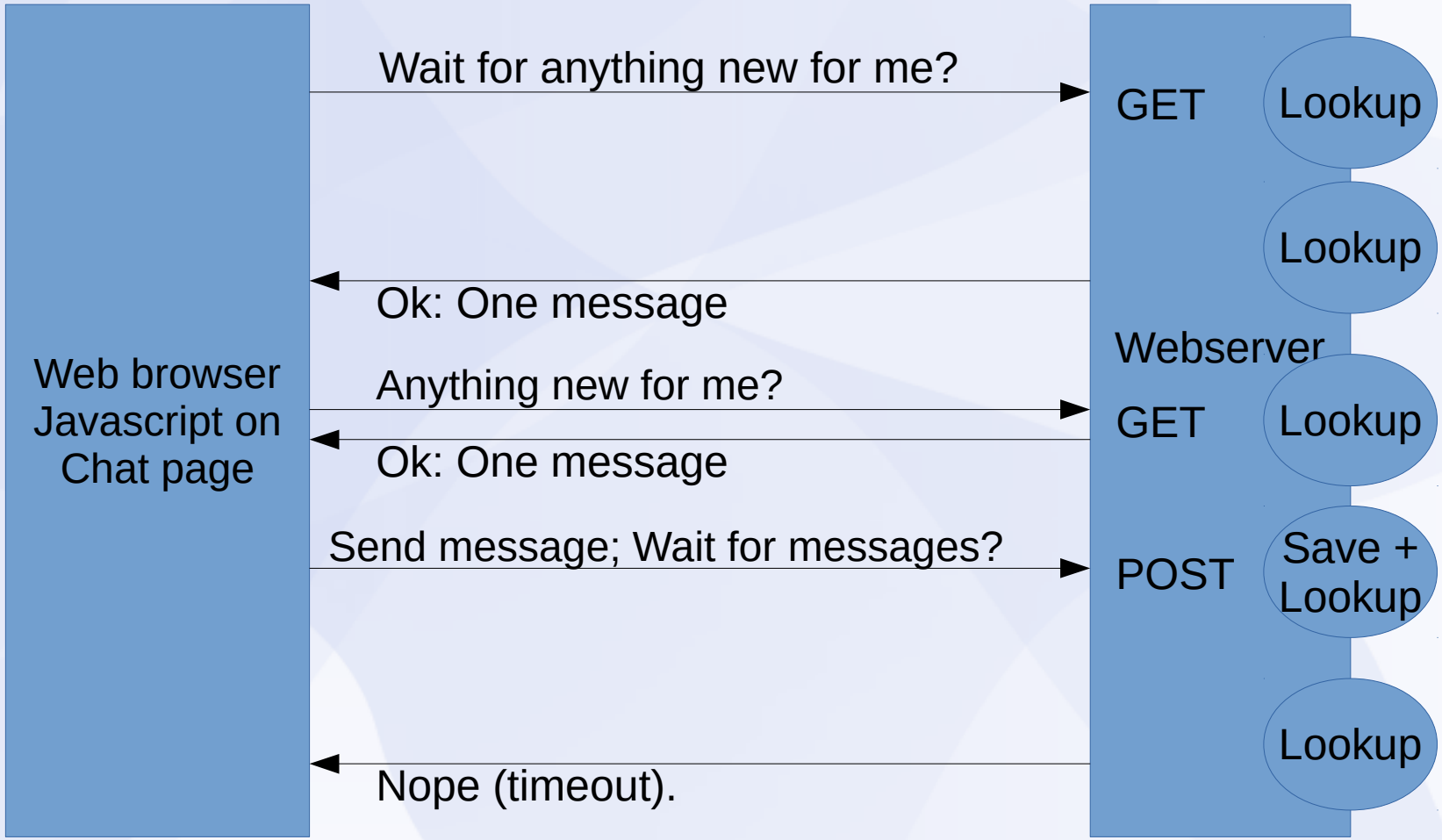
Before websockets



5 requests/responses (tcp+http+cookies etc)

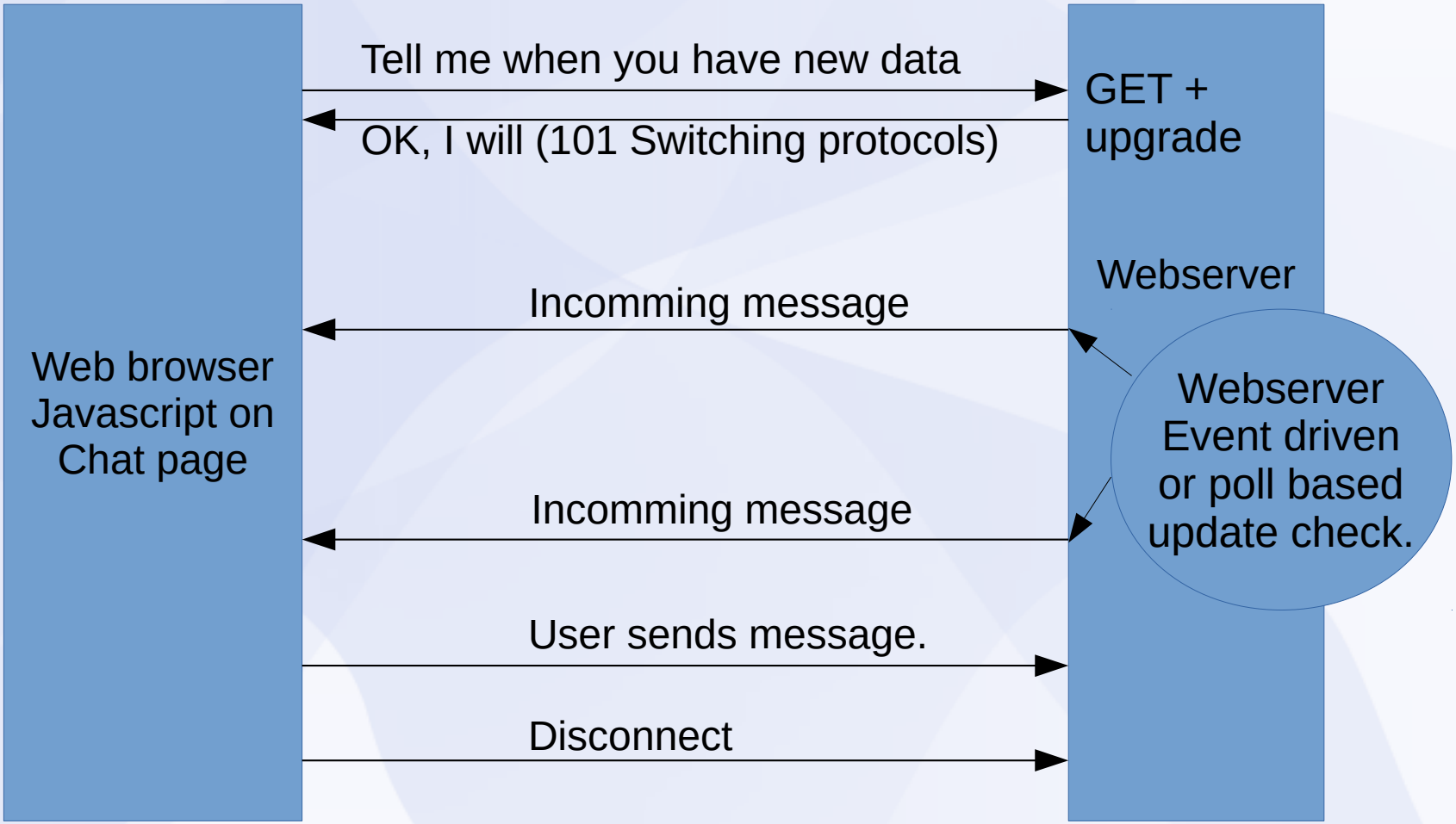
Long Poll

Before websockets



3 requests/responses (tcp+http+cookies etc)

Full duplex
With websockets



1 request/response (tcp+http+cookies etc) + 4 short messages

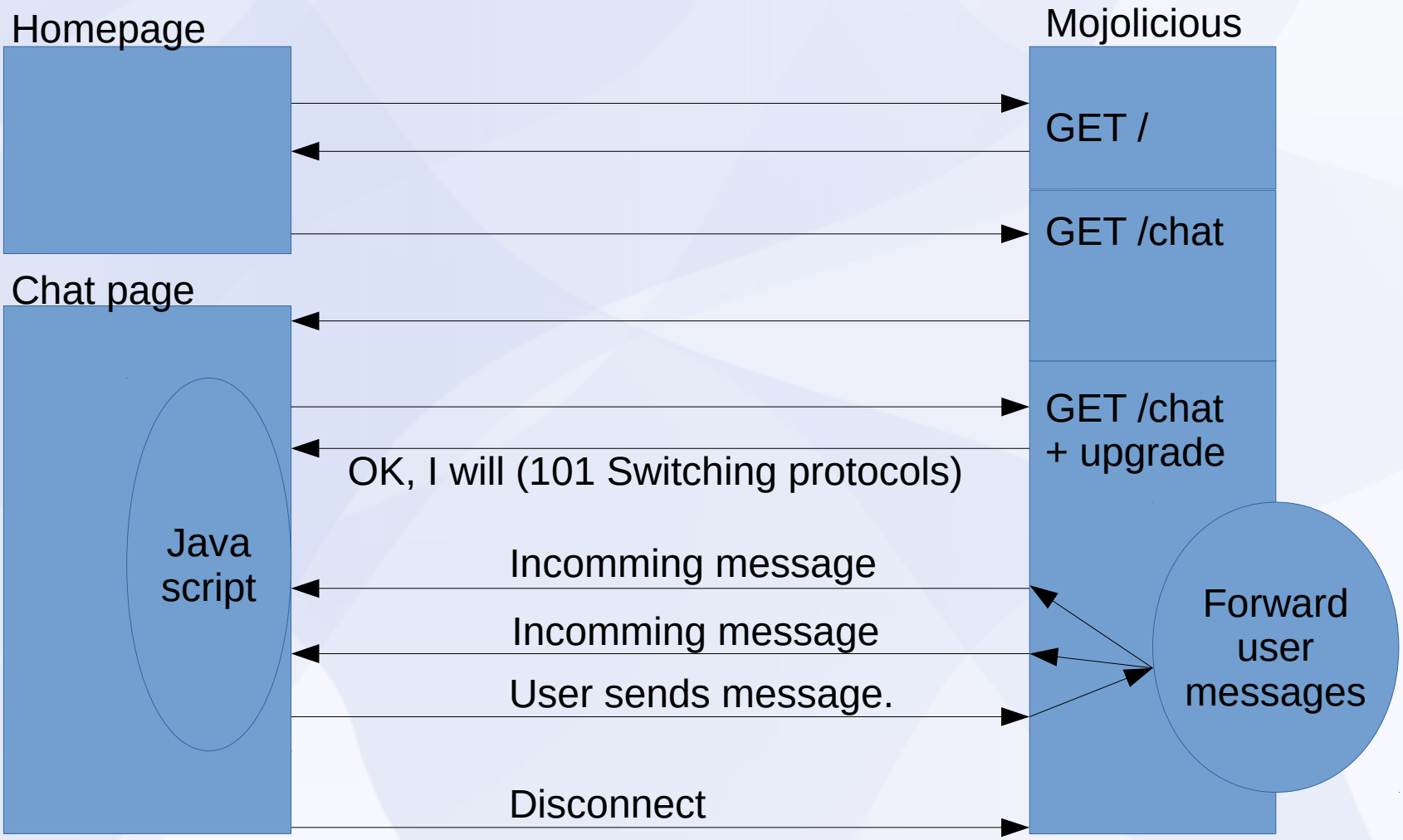
Websockets
long polling and websockets

If connection is lost,
you may want to reconnect.

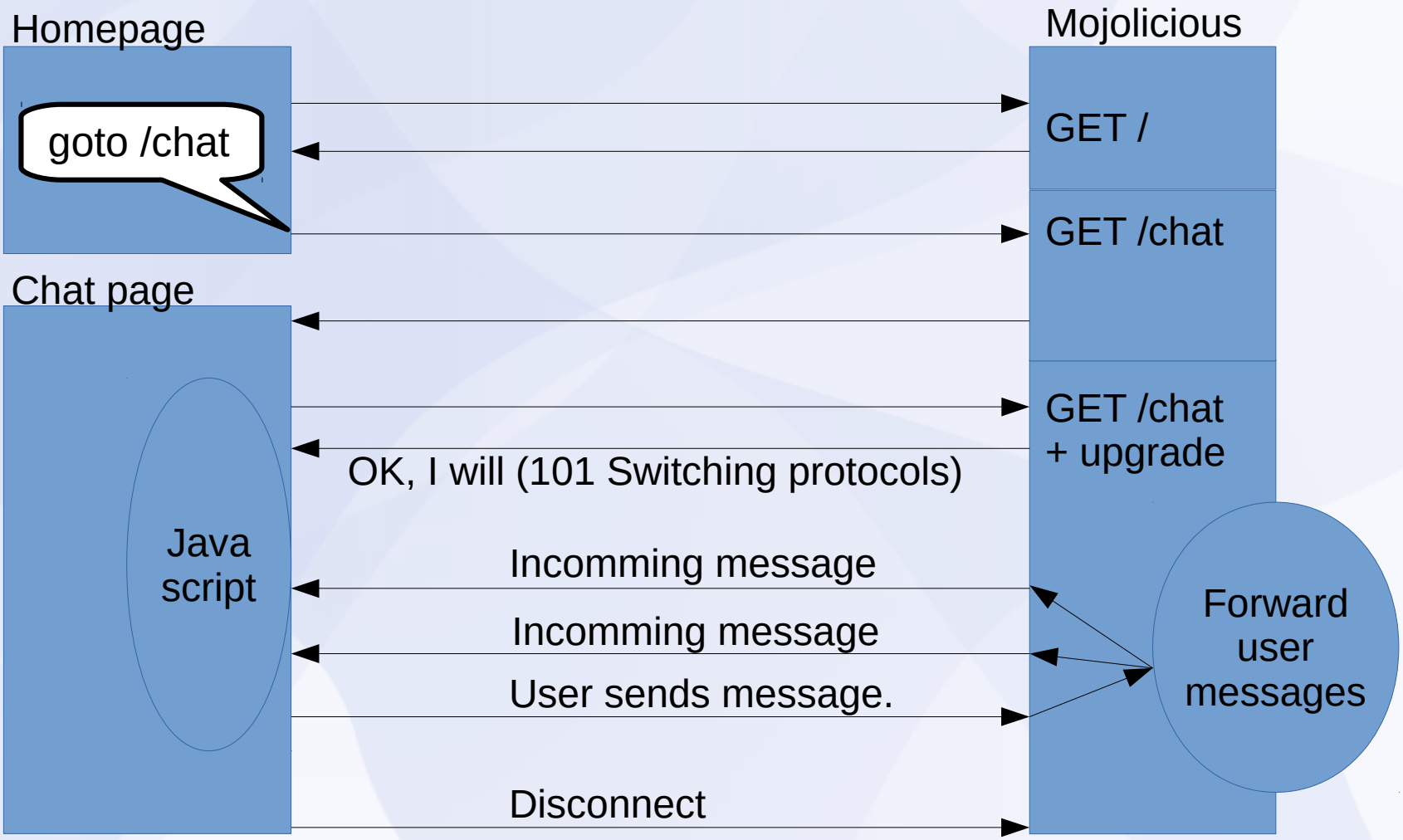
But make sure
to add a backoff algorithm
in order to reduce load
on the server and network.

Read also: http://blog.johnryding.com/post/78544969349/how-to-reconnect-web-sockets-in-a-realtime-web-app
or search for Exponential Backoff algorithm.

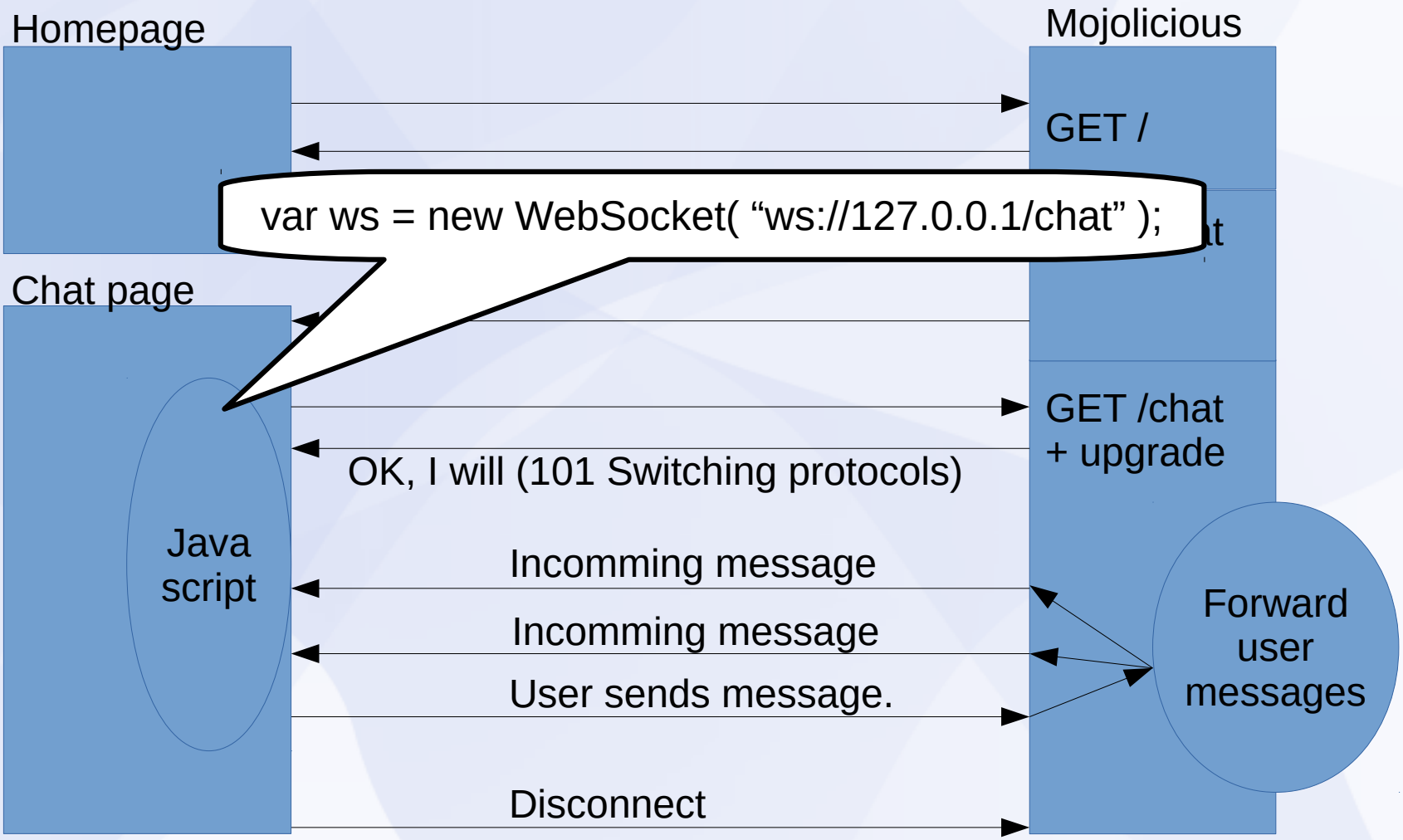
Our setup
With websockets



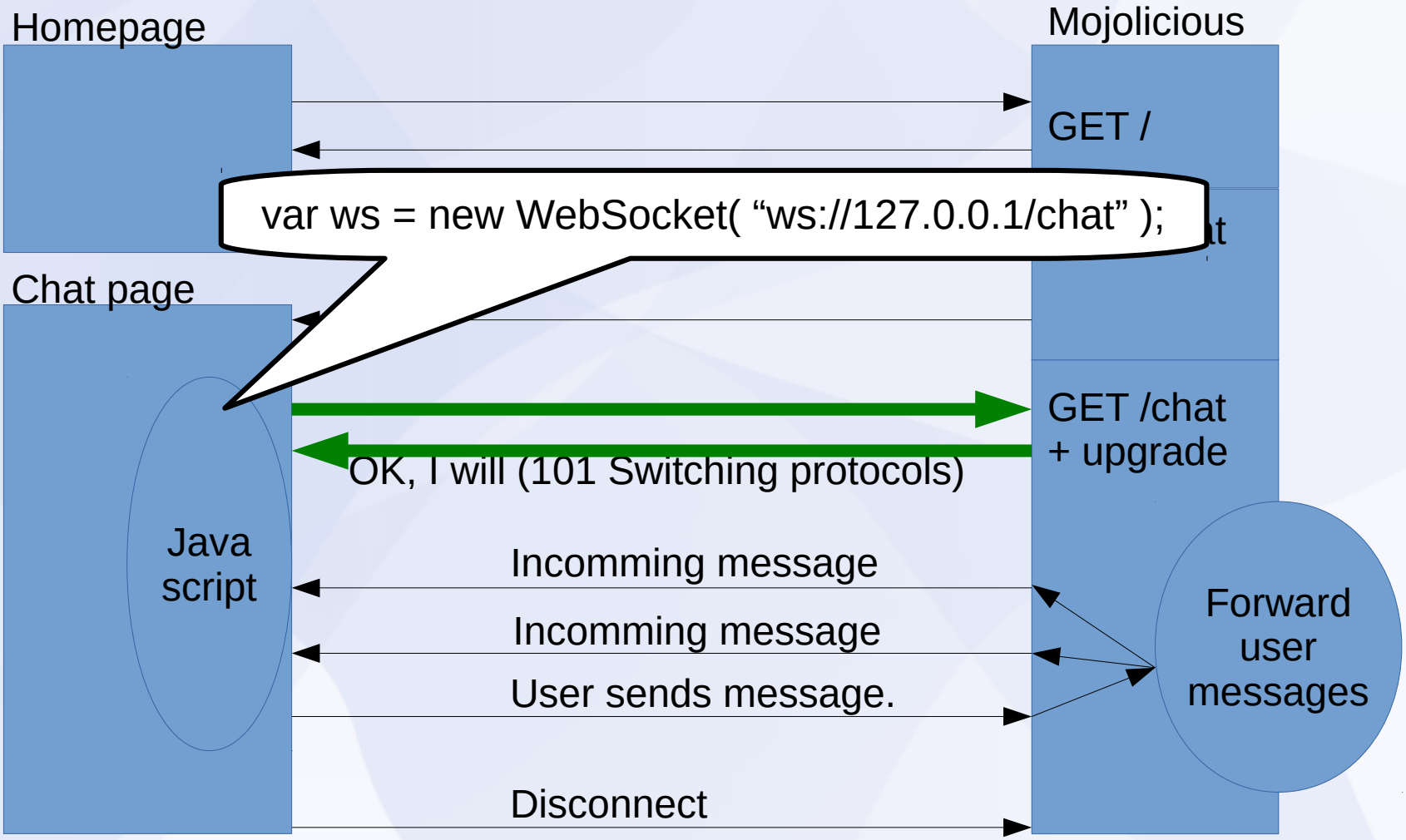
Our setup
With websockets



Our setup
With websockets

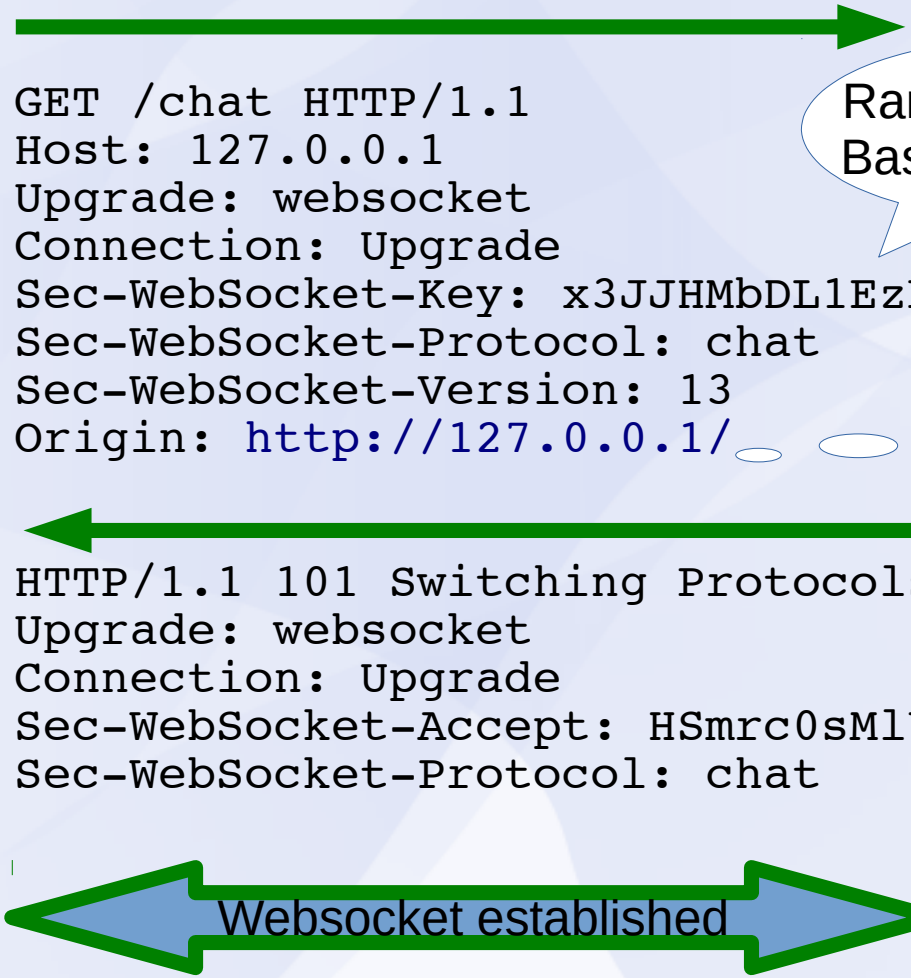


Our setup
With websockets

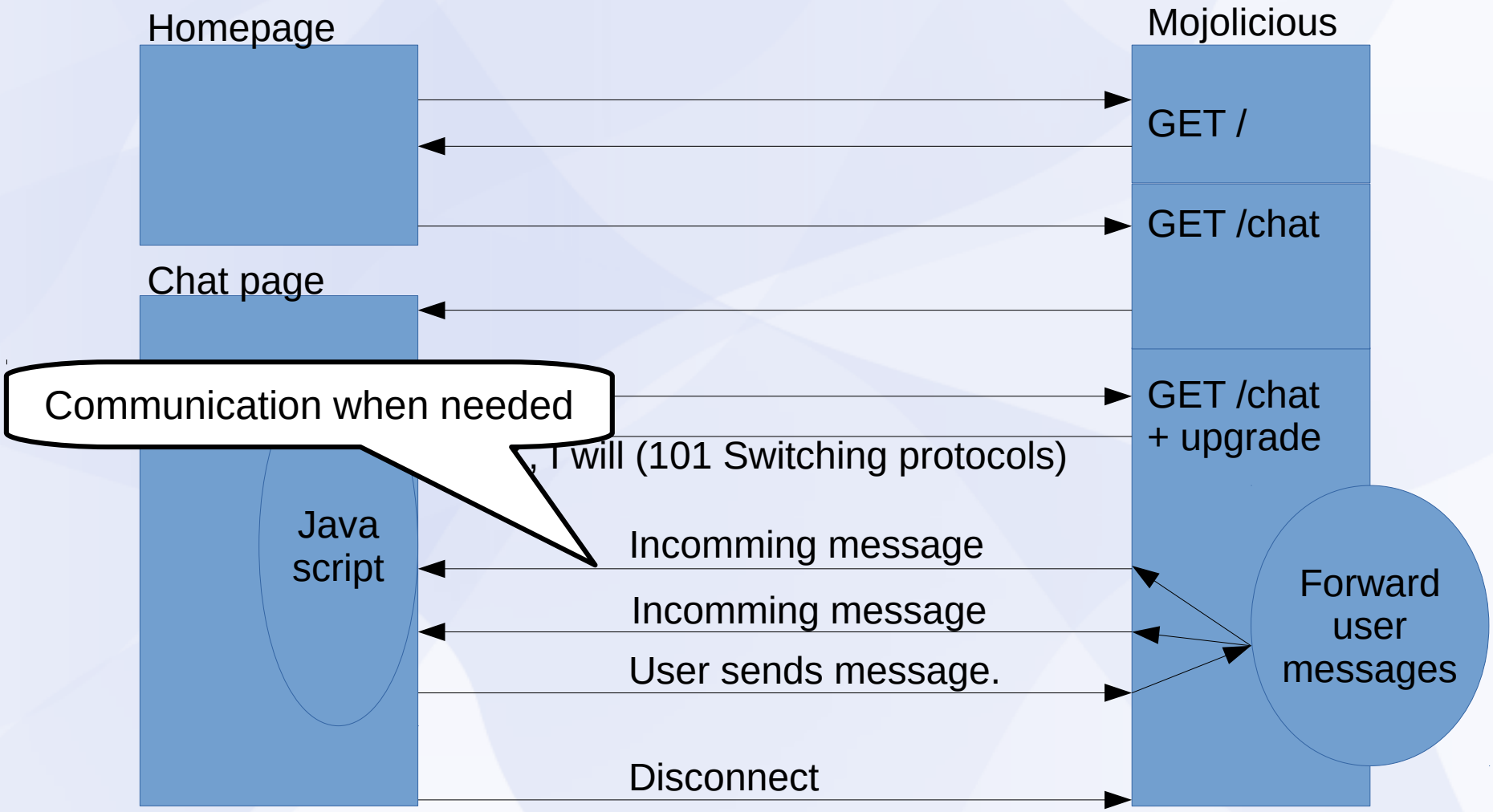


Websockets

Connection setup



Our setup
With websockets



Examples

Examples

- Create new Mojolicious app.

a) Send message, every 3 sec.

b) Chat script – multiple clients talking together.

- Application testing.

Examples

- Create new Mojolicious app.
  http://127.0.0.1/

a) Send message, every 3 sec.
  http://127.0.0.1/time

b) Chat script – multiple clients talking together.
  http://127.0.0.1/chat

- Application testing.

Examples

- Create new Mojolicious app.
  http://127.0.0.1/
  Home page, Wswork.pm, template

a) Send message, every 3 sec.
  http://127.0.0.1/time

b) Chat script – multiple clients talking together.
  http://127.0.0.1/chat

- Application testing.

Create new Mojolicious app

```
$ sudo cpan Mojolicious

$ mojo generate app Wstest


$ cd wstest

$ touch log/development.log
$ tail -f log/development.log &

$ morbo script/wstest
```

Examples

- Create new Mojolicious app.
  http://127.0.0.1/
  Home page, Wswork.pm, template

a) Send message, every 3 sec.
  http://127.0.0.1/time
  Base page, Wstime.pm, template, websocket, javascript

b) Chat script – multiple clients talking together.
  http://127.0.0.1/chat

- Application testing.

Examples

- Create new Mojolicious app.
  http://127.0.0.1/
  Home page, Wswork.pm, template

a) Send message, every 3 sec.
  http://127.0.0.1/time
  Base page, Wstime.pm, template, websocket, javascript

b) Chat script – multiple clients talking together.
  http://127.0.0.1/chat
  Base page, Wschat.pm, template, websocket, javascript

- Application testing.

lib/Wstest.pm

```
package Wstest;
use Mojo::Base 'Mojolicious';

# This method will run once at server start
sub startup {
  my $self = shift;

  # Router
  my $r = $self->routes;

  #
  # Top index page
  #
  $r->get('/')->to('wswork#welcome');

  #
  # /time  -- Websocket and GET welcome page
  #
  $r->websocket('/time')->to('wstime#accept' );
  $r->get(      '/time')->to('wstime#welcome');

}

1;
```

lib/Wstest.pm

```
package Wstest;
use Mojo::Base 'Mojolicious';

# This method will run once at server start
sub startup {
  my $self = shift;

  # Router
  my $r = $self->routes;

  #
  # Top index page
  #
  $r->get('/')->to('wswork#welcome');

  #
  # /time  -- Websocket and GET welcome page
  #
  $r->websocket('/time')->to('wstime#accept' );
  $r->get(      '/time')->to('wstime#welcome');

}

1;
```

lib/Wstest.pm

```
package Wstest;
use Mojo::Base 'Mojolicious';

# This method will run once at server start
sub startup {
  my $self = shift;

  # Router
  my $r = $self->routes;

  #
  # Top index page
  #
  $r->get('/')->to('wswork#w          ');

  #
  # /time  -- Websocket and GET welcome page
  #
  $r->websocket('/time')->to('wstime#accept' );
  $r->get(      '/time')->to('wstime#welcome');

}

1;
```

If you use same URL,
make sure that the websocket comes **first**,

since a websocket is essentially
an upgraded GET request.

lib/Wstest/Wswork.pm
for http://127.0.0.1/

```
package Wstest::Wswork;

use Mojo::Base 'Mojolicious::Controller';

sub welcome {
}

1;
```

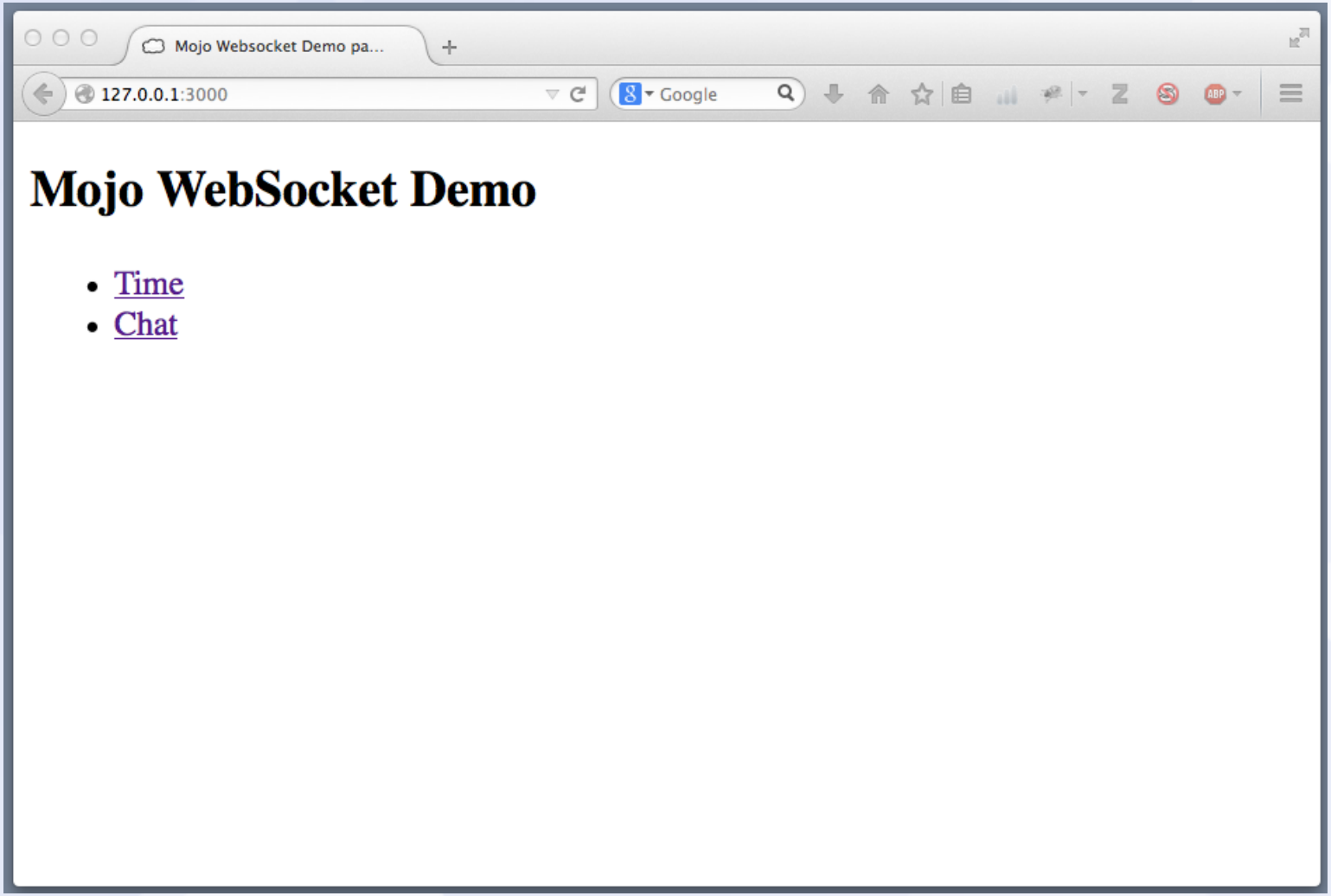
templates/wswork/welcome.html.ep
for http://127.0.0.1/

```
<!doctype html>
<html>
<head>
    <title>Mojo Websocket Demo pages</title>

    <link rel="stylesheet" type="text/css"
href="/css/mystyle.css" />
</head>
<body>
    <h2>Mojo WebSocket Demo</h2>

    <ul>
     <li> <a href="/time">Time</a> </li>
     <li> <a href="/chat">Chat</a> </li>
    </ul>
</body>
</html>
```

http://127.0.0.1:3000/



Examples

- Create new Mojolicious app.

a) Send message, every 3 sec.

b) Chat script – multiple clients talking together.

- Application testing.

templates/wstime/welcome.html.ep
for http://127.0.0.1/time/

```
<!doctype html>
<html>
<head>
    <title>Mojo Websocket Time Demo</title>

    <link rel="stylesheet" type="text/css"
          href="/css/mystyle.css" />

    <script type="text/javascript"
             src="/js/jquery.js"></script>
    <script type="text/javascript"
             src="/js/wstime.js"></script>

</head>
<body>
    <h2>Mojo WebSocket Demo</h2>
    <div>
      <div class="timearea">
        ## Data should appear here
      </div>
    </div>
</body>
</html>
```

templates/wstime/welcome.html.ep
for http://127.0.0.1/time/

```
<!doctype html>
<html>
<head>
    <title>Mojo Websocket Time Demo</title>

    <link rel="stylesheet" type="text/css"
          href="/css/mystyle.css" />

    <script type="text/javascript"
             src="/js/jquery.js"></script>
    <script type="text/javascript"
             src="/js/wstime.js"></script>

</head>
<body>
    <h2>Mojo WebSocket Demo</h2>
    <div>
      <div class="timearea">
        ## Data should appear here
      </div>
    </div>
</body>
</html>
```

/js/wstime.js
=
public/js/wstime.js

public/js/wstime.js
for http://127.0.0.1/js/wstime.js

```
// Take the current document location href
// and replace http or https with ws or wss

console.log("url  :  " + document.location.href );

var wsurl = document.location.href.replace(/^http/i,"ws");

console.log("wsurl:  " + wsurl );

...
```

public/js/wstime.js
for http://127.0.0.1/js/wstime.js

```
// Take the current document location href
// and replace http or https with ws or wss

console.log("url  :  " + document.location.href );

var wsurl = document.location.href.replace(/^http/i,"ws");

console.log("wsurl:  " + wsurl );

...
```

http://127.0.0.1:3000/time/
=>
ws://127.0.0.1:3000/time/

public/js/wstime.js
for http://127.0.0.1/js/wstime.js

```
// Take the current document location href
// and replace http or https with ws or wss

console.log("url  :  " + document.location.href );

var wsurl = document.location.href.replace(/^http/i,"ws");

console.log("wsurl:  " + wsurl );

...
```

https://127.0.0.1:3000/time/
=>
wss://127.0.0.1:3000/time/

public/js/wstime.js
for http://127.0.0.1/js/wstime.js

```
// Take the current document location href
// and replace http or https with ws or wss

console.log("url  :  " + document.location.href );

var wsurl = document.location.href.replace(/^http/i,"ws");

console.log("wsurl:  " + wsurl );

...
```

public/js/wstime.js
for http://127.0.0.1/js/wstime.js

```
...

console.log("About to connect to websocket on: " + wsurl);

var ws = new WebSocket( wsurl );

ws.onopen = function() {
    console.log("Websocket open");
};

ws.onmessage = function(msg) {
    console.log("Got message: "+ msg);

    // var res = JSON.parse(msg.data);

    // Lets do something with the message
    $('.timearea').append( "<br>" +
        $("<div/>").text(msg.data).html()
    );
}
```

public/js/wstime.js
for http://127.0.0.1/js/wstime.js

```
...

console.log("About to connect to websocket on: " + wsurl);

var ws = new WebSocket( wsurl );

ws.onopen = function() {
    console.log("Websocket open");
};

ws.onmessage = function(msg) {
    console.log("Got message: "+ msg);

    // var res = JSON.parse(msg.data);

    // Lets do something with the message
    $('.timearea').append( "<br>" +
        $("<div/>").text(msg.data).html()
    );
}
```

public/js/wstime.js
for http://127.0.0.1/js/wstime.js

```
...

console.log("About to connect to websocket on: " + wsurl);

var ws = new WebSocket( wsurl );

ws.onopen = function() {
    console.log("Websocket open");
};

ws.onmessage = function(msg) {
    console.log("Got message: "+ msg);

    // var res = JSON.parse(msg.data);

    // Lets do something with the message
    $('.timearea').append( "<br>" +
        $("<div/>").text(msg.data).html()
    );
}
```

public/js/wstime.js
for http://127.0.0.1/js/wstime.js

```
...

console.log("About to connect to websocket on: " + wsurl);

var ws = new WebSocket( wsurl );

ws.onopen = function() {
    console.log("Websocket open");
};

ws.onmessage = function(msg) {
    console.log("Got message: "+ msg);

    // var res = JSON.parse(msg.data);

    // Lets do something with the message
    $('.timearea').append( "<br>" +
        $("<div/>").text(msg.data).html()
    );
}
```

public/js/wstime.js
for http://127.0.0.1/js/wstime.js

```
...

console.log("About to connect to websocket on: " + wsurl);

var ws = new WebSocket( wsurl );

ws.onopen = function() {
    console.log("Websocket open");
};

ws.onmessage = function(msg) {
    console.log("Got message: "+ msg);

    // var res = JSON.parse(msg.data);

    // Lets do something with the message
    $('.timearea').append( "<br>" +
        $("<div/>").text(msg.data).html()
    );
}
```

public/js/wstime.js
for http://127.0.0.1/js/wstime.js

```
...

console.log("About to connect to websocket on: " + wsurl);

var ws = new WebSocket( wsurl );

ws.onopen = function() {
    console.log("Websocket open");
};

ws.onmessage = function(msg) {
    console.log("Got message: "+ msg);

    // var res = JSON.parse(msg.data);

    // Lets do something with the message
    $('.timearea').append( "<br>" +
        $("<div/>").text(msg.data).html()
    );
}
```

Websocket funtionality
from javascript

```
var ws = new WebSocket( 'ws://some.host/path' );


ws.onopen    = function() {};        On opening socket
ws.onclose   = function() {};        On closing socket

ws.onmessage = function(msg) {};     On socket message

ws.onerror   = function() {};        On broken connection


ws.send("Some message");     Send message.

ws.close();                  Close socket.
ws.destroy();                Destroy socket and free res.
```

Websocket funtionality
from javascript

```
var ws = new WebSocket( 'ws://some.host/path' );


ws.onopen    = function() {};        On opening socket
ws.onclose   = function() {};        On closing socket

ws.onmessage = function(msg) {};     On socket message

ws.onerror   = function() {};        On broken connection


ws.send("Some message");     Send message.

ws.close();                  Close socket.
ws.destroy();                Destroy socket and free res.
```

Websocket funtionality
from javascript

```
var ws = new WebSocket( 'ws://some.host/path' );


ws.onopen    = function() {};        On opening socket
ws.onclose   = function() {};        On closing socket

ws.onmessage = function(msg) {};     On socket message

ws.onerror   = function() {};        On broken connection


ws.send("Some message");     Send message.

ws.close();                  Close socket.
ws.destroy();                Destroy socket and free res.
```

Websocket funtionality
from javascript

```
var ws = new WebSocket( 'ws://some.host/path' );


ws.onopen    = function() {};        On opening socket
ws.onclose   = function() {};        On closing socket

ws.onmessage = function(msg) {};     On socket message

ws.onerror   = function() {};        On broken connection


ws.send("Some message");     Send message.

ws.close();                  Close socket.
ws.destroy();                Destroy socket and free res.
```

Websocket funtionality
from javascript

```
var ws = new WebSocket( 'ws://some.host/path' );


ws.onopen    = function() {};        On opening socket
ws.onclose   = function() {};        On closing socket

ws.onmessage = function(msg) {};     On socket message

ws.onerror   = function() {};        On broken connection


ws.send("Some message");     Send message.

ws.close();                  Close socket.
ws.destroy();                Destroy socket and free res.
```

Websocket funtionality
from javascript

```
var ws = new WebSocket( 'ws://some.host/path' );


ws.onopen    = function() {};        On opening socket
ws.onclose   = function() {};        On closing socket

ws.onmessage = function(msg) {};     On socket message

ws.onerror   = function() {};        On broken connection


ws.send("Some message");     Send message.

ws.close();                  Close socket.
ws.destroy();                Destroy socket and free res.
```

Websocket funtionality
from javascript

```
var ws = new WebSocket( 'ws://some.host/path' );


ws.onopen    = function() {};        On opening socket
ws.onclose   = function() {};        On closing socket

ws.onmessage = function(msg) {};     On socket message

ws.onerror   = function() {};        On broken connection


ws.send("Some message");     Send message.

ws.close();                  Close socket.
ws.destroy();                Destroy socket and free res.
```

Websocket funtionality
from javascript

```
var ws = new WebSocket( 'ws://some.host/path' );


ws.onopen    = function() {};        On opening socket
ws.onclose   = function() {};        On closing socket

ws.onmessage = function(msg) {};     On socket message

ws.onerror   = function() {};        On broken connection


ws.send("Some message");     Send message.

ws.close();                  Close socket.
ws.destroy();                Destroy socket and free res.
```

Websocket funtionality
from javascript

```
var ws = new WebSocket( 'ws://some.host/path' );


ws.onopen    = function() {};        On opening socket
ws.onclose   = function() {};        On closing socket

ws.onmessage = function(msg) {};     On socket message

ws.onerror   = function() {};        On broken connection


ws.send("Some message");     Send message.

ws.close();                  Close socket.
ws.destroy();                Destroy socket and free res.
```

Websocket funtionality
from javascript

```
var ws = new WebSocket( 'ws://some.host/path' );


ws.onopen    = function() {};        On opening socket
ws.onclose   = function() {};        On closing socket

ws.onmessage = function(msg) {};     On socket message

ws.onerror   = function() {};        On broken connection


ws.send("Some message");     Send message.

ws.close();                  Close socket.
ws.destroy();                Destroy socket and free res.
```

Websocket funtionality
from javascript

```
var ws = new WebSocket( 'ws://some.host/path' );

if (ws.readyState === ws.OPEN)
{
    // Connection is ready for sending data
}
else if (ws.readyState === ws.CLOSED)
{
    // Connection was closed
}
```

Constant	Value	Description
CONNECTING	0	The connection is not yet open.
OPEN	1	The connection is open and ready to communicate.
CLOSING	2	The connection is in the process of closing.
CLOSED	3	The connection is closed or couldn't be opened.

lib/Wstest/Wstime.pm

```
package Wstest::Wstime;
use Mojo::Base 'Mojolicious::Controller';

sub welcome {
    # default renders templates/wstime/welcome.html.ep
}

sub accept {
    # ... see next slide ...
}

1;
```

lib/Wstest/Wstime.pm

```
sub accept {
    my $self = shift;

    my $interval = 3;

    # How long to go without events before closing connection.
    # Logs timeouts as: "Inactivity timeout"
    Mojo::IOLoop->stream($self->tx->connection)->timeout(300);

    my $scheduler = Mojo::IOLoop->recurring( $interval => sub {

        my $data = scalar localtime();

        $self->app->log->debug("Sending data: ".$data);
        $self->send( $data );  # Possibly add JSON encoding

    });

    $self->on( finish => sub {
        $self->app->log->debug("Finished websocket");
        Mojo::IOLoop->remove( $scheduler );
    });

}
```

lib/Wstest/Wstime.pm

```
sub accept {
    my $self = shift;

    my $interval = 3;

    # How long to go without events before closing connection.
    # Logs timeouts as: "Inactivity timeout"
    Mojo::IOLoop->stream($self->tx->connection)->timeout(300);

    my $scheduler = Mojo::IOLoop->recurring( $interval => sub {

        my $data = scalar localtime();

        $self->app->log->debug("Sending data: ".$data);
        $self->send( $data );  # Possibly add JSON encoding

    });

    $self->on( finish => sub {
        $self->app->log->debug("Finished websocket");
        Mojo::IOLoop->remove( $scheduler );
    });

}
```

lib/Wstest/Wstime.pm

```
sub accept {
    my $self = shift;

    my $interval = 3;

    # How long to go without events before closing connection.
    # Logs timeouts as: "Inactivity timeout"
    Mojo::IOLoop->stream($self->tx->connection)->timeout(300);

    my $scheduler = Mojo::IOLoop->recurring( $interval => sub {

        my $data = scalar localtime();

        $self->app->log->debug("Sending data: ".$data);
        $self->send( $data );  # Possibly add JSON encoding

    });

    $self->on( finish => sub {
        $self->app->log->debug("Finished websocket");
        Mojo::IOLoop->remove( $scheduler );
    });

}
```

lib/Wstest/Wstime.pm

```
sub accept {
    my $self = shift;

    my $interval = 3;

    # How long to go without events before closing connection.
    # Logs timeouts as: "Inactivity timeout"
    Mojo::IOLoop->stream($self->tx->connection)->timeout(300);

    my $scheduler = Mojo::IOLoop->recurring( $interval => sub {

        my $data = scalar localtime();

        $self->app->log->debug("Sending data: ".$data);
        $self->send( $data );  # Possibly add JSON encoding

    });

    $self->on( finish => sub {
        $self->app->log->debug("Finished websocket");
        Mojo::IOLoop->remove( $scheduler );
    });

}
```

lib/Wstest/Wstime.pm

```
sub accept {
    my $self = shift;

    my $interval = 3;

    # How long to go without events before closing connection.
    # Logs timeouts as: "Inactivity timeout"
    Mojo::IOLoop->stream($self->tx->connection)->timeout(300);

    my $scheduler = Mojo::IOLoop->recurring( $interval => sub {

        my $data = scalar localtime();

        $self->app->log->debug("Sending data: ".$data);
        $self->send( $data );  # Possibly add JSON encoding

    });

    $self->on( finish => sub {
        $self->app->log->debug("Finished websocket");
        Mojo::IOLoop->remove( $scheduler );
    });

}
```

lib/Wstest/Wstime.pm

```
sub accept {
    my $self = shift;

    my $interval = 3;

    # How long to go without events before closing connection.
    # Logs timeouts as: "Inactivity timeout"
    Mojo::IOLoop->stream($self->tx->connection)->timeout(300);

    my $scheduler = Mojo::IOLoop->recurring( $interval => sub {

        my $data = scalar localtime();

        $self->app->log->debug("Sending data: ".$data);
        $self->send( $data );  # Possibly add JSON encoding

    });

    $self->on( finish => sub {
        $self->app->log->debug("Finished websocket");
        Mojo::IOLoop->remove( $scheduler );
    });

}
```

lib/Wstest/Wstime.pm

```
sub accept {
    my $self = shift;

    my $interval = 3;

    # How long to go without events before closing connection.
    # Logs timeouts as: "Inactivity timeout"
    Mojo::IOLoop->stream($self->tx->connection)->timeout(300);

    my $scheduler = Mojo::IOLoop->recurring( $interval => sub {

        my $data = scalar localtime();

        $self->app->log->debug("Sending data: ".$data);
        $self->send( $data );  # Possibly add JSON encoding

    });

    $self->on( finish => sub {
        $self->app->log->debug("Finished websocket");
        Mojo::IOLoop->remove( $scheduler );
    });

}
```

lib/Wstest/Wstime.pm

```
sub accept {
    my $self = shift;

    my $interval = 3;

    # How long to go without events before closing connection.
    # Logs timeouts as: "Inactivity timeout"
    Mojo::IOLoop->stream($self->tx->connection)->timeout(300);

    my $scheduler = Mojo::IOLoop->recurring( $interval => sub {

        my $data = scalar localtime();

        $self->app->log->debug("Sending data: ".$data);
        $self->send( $data );  # Possibly add JSON encoding

    });

    $self->on( finish => sub {
        $self->app->log->debug("Finished websocket");
        Mojo::IOLoop->remove( $scheduler );
    });

}
```

Demo

Examples

- Create new Mojolicious app.

a) Send message, every 3 sec.

b) Chat script – multiple clients talking together.

- Application testing.

lib/Wstest.pm

```
package Wstest;
use Mojo::Base 'Mojolicious';

# This method will run once at server start
sub startup {

  …

  #
  # /time  -- Websocket and GET welcome page
  #
  $r->websocket('/time')->to('wstime#accept');
  $r->get(       '/time')->to('wstime#welcome');









}

1;
```

lib/Wstest.pm

```
package Wstest;
use Mojo::Base 'Mojolicious';

# This method will run once at server start
sub startup {

  …

  #
  # /time  -- Websocket and GET welcome page
  #
  $r->websocket('/time')->to('wstime#accept');
  $r->get(       '/time')->to('wstime#welcome');

  #
  # /chat  -- Websocket and GET welcome page
  #
  $r->websocket('/chat')->to('wschat#accept');
  $r->get(       '/chat')->to('wschat#welcome');

}

1;
```

templates/wschat/welcome.html.ep
for http://127.0.0.1/chat/

```
<!doctype html>
<html>
<head>
   <title>Mojo Websocket Demo pages</title>

   <link rel="stylesheet" type="text/css"
        href="/css/mystyle.css" />
   <script type="text/javascript" src="/js/jquery.js"></script>

</head>
<body>
   <h2>Mojo WebSocket Demo</h2>
   <div>
     <form>
     <div class="chatarea">
       ## Data here
     </div>
     <input class="chatinput" type="text" value="">
     </form>
   </div>
</body>
   <script type="text/javascript" src="/js/wschat.js"></script>
</html>
```

templates/wschat/welcome.html.ep
for http://127.0.0.1/chat/

```
<!doctype html>
<html>
<head>
   <title>Mojo Websocket Demo pages</title>

   <link rel="stylesheet" type="text/css"
        href="/css/mystyle.css" />
   <script type="text/javascript" src="/js/jquery.js"></script>

</head>
<body>
   <h2>Mojo WebSocket Demo</h2>
   <div>
     <form>
     <div class="chatarea">
       ## Data here
     </div>
     <input class="chatinput" type="text" value="">
     </form>
   </div>
</body>
   <script type="text/javascript" src="/js/wschat.js"></script>
</html>
```

templates/wschat/welcome.html.ep
for http://127.0.0.1/chat/

```
<!doctype html>
<html>
<head>
   <title>Mojo Websocket Demo pages</title>

   <link rel="stylesheet" type="text/css"
        href="/css/mystyle.css" />
   <script type="text/javascript" src="/js/jquery.js"></script>

</head>
<body>
   <h2>Mojo WebSocket Demo</h2>
   <div>
     <form>
     <div class="chatarea">
       ## Data here
     </div>
     <input class="chatinput" type="text" value="">
     </form>
   </div>
</body>
   <script type="text/javascript" src="/js/wschat.js"></script>
</html>
```

templates/wschat/welcome.html.ep
for http://127.0.0.1/chat/

```
<!doctype html>
<html>
<head>
   <title>Mojo Websocket Demo pages</title>

   <link rel="stylesheet" type="text/css"
        href="/css/mystyle.css" />
   <script type="text/javascript" src="/js/jquery.js"></script>

</head>
<body>
   <h2>Mojo WebSocket Demo</h2>
   <div>
     <form>
     <div class="chatarea">
       ## Data here
     </div>
     <input class="chatinput" type="text" value="">
     </form>
   </div>
</body>
   <script type="text/javascript" src="/js/wschat.js"></script>
</html>
```

public/js/wschat.js

```
var wsurl = document.location.href.replace(/^http/i,"ws");
console.log("About to connect to websocket on: " + wsurl);

var ws = new WebSocket( wsurl );

ws.onopen = function() {
    console.log("Websocket open");
};

ws.onmessage = function(msg) {
    console.log("Got message: ", msg);

    // var res = JSON.parse(msg.data);

    // Lets do something with the message
    $('.chatarea').append("<br>" +
        $("<div/>").text(msg.data).html()
    );

    $('.chatinput').focus();
}

...
```

public/js/wschat.js

```
var wsurl = document.location.href.replace(/^http/i,"ws");
console.log("About to connect to websocket on: " + wsurl);

var ws = new WebSocket( wsurl );

ws.onopen = function() {
    console.log("Websocket open");
};

ws.onmessage = function(msg) {
    console.log("Got message: ", msg);

    // var res = JSON.parse(msg.data);

    // Lets do something with the message
    $('.chatarea').append("<br>" +
        $("<div/>").text(msg.data).html()
    );

    $('.chatinput').focus();
}

...
```

public/js/wschat.js

```
var wsurl = document.location.href.replace(/^http/i,"ws");
console.log("About to connect to websocket on: " + wsurl);

var ws = new WebSocket( wsurl );

ws.onopen = function() {
    console.log("Websocket open");
};

ws.onmessage = function(msg) {
    console.log("Got message: ", msg);

    // var res = JSON.parse(msg.data);

    // Lets do something with the message
    $('.chatarea').append("<br>" +
        $("<div/>").text(msg.data).html()
    );

    $('.chatinput').focus();
}

...
```

public/js/wschat.js

```
...

$('.chatinput').keydown( function(e) {
  if ( e.which != 13 ) return true;

  var inp = $('.chatinput');
  var msg = inp.val();
  console.log("Send message: "+msg);
  ws.send(msg);
  inp.val('');
  return false; // Do not use keypress for anything
});
```

public/js/wschat.js

```
...

$('.chatinput').keydown( function(e) {
  if ( e.which != 13 ) return true;

  var inp = $('.chatinput');
  var msg = inp.val();
  console.log("Send message: "+msg);
  ws.send(msg);
  inp.val('');
  return false; // Do not use keypress for anything
});
```

public/js/wschat.js

```
...

$('.chatinput').keydown( function(e) {
  if ( e.which != 13 ) return true;

  var inp = $('.chatinput');
  var msg = inp.val();
  console.log("Send message: "+msg);
  ws.send(msg);
  inp.val('');
  return false; // Do not use keypress for anything
});
```

public/js/wschat.js

```
...

$('.chatinput').keydown( function(e) {
  if ( e.which != 13 ) return true;

  var inp = $('.chatinput');
  var msg = inp.val();
  console.log("Send message: "+msg);
  ws.send(msg);
  inp.val('');
  return false; // Do not use keypress for anything
});
```

public/js/wschat.js

```
...

$('.chatinput').keydown( function(e) {
  if ( e.which != 13 ) return true;

  var inp = $('.chatinput');
  var msg = inp.val();
  console.log("Send message: "+msg);
  ws.send(msg);
  inp.val('');
  return false; // Do not use keypress for anything
});
```

public/js/wschat.js

```
...

$('.chatinput').keydown( function(e) {
  if ( e.which != 13 ) return true;

  var inp = $('.chatinput');
  var msg = inp.val();
  console.log("Send message: "+msg);
  ws.send(msg);
  inp.val('');
  return false; // Do not use keypress for anything
});
```

lib/Wstest/Wschat.pm

```
package Wstest::Wschat;
use Mojo::Base 'Mojolicious::Controller';

sub welcome {
}

my %client_ws;
my %client_alias;

sub accept {
   ...
}

sub broadcast {
   ...
}

1;
```

lib/Wstest/Wschat.pm
sub accept

```
sub accept {
    my $self = shift;

    # How long to go without events before closing connection.
    # Logs timeouts as: "Inactivity timeout"
    Mojo::IOLoop->stream($self->tx->connection)->timeout(300);

    # $self->tx stringifies to something like
    #  "Mojo::Transaction::WebSocket=HASH(0x7fed310840b0)"
    # We change this for aliasing

    my $id = $self->tx =~ /(0x\w+)/ && $1;

    ...
```

lib/Wstest/Wschat.pm
sub accept

```
    …

    if ( my @names = values %client_alias ) {
        $self->send("## Talking to: $_")  foreach @names;
    } else {
        $self->send("## Nobody else to talk to at the moment...");
    }
    $self->send("## You are known as $id.");
    $self->send("## Please send your name as the first message.");

    $self->broadcast("## ${id} joined the conversation.");
    $client_ws{ $id } = $self;
    $client_alias{ $id } = $id;

    $self->app->log->debug(scalar(keys %client_ws)." Active
clients.");

    ...
```

lib/Wstest/Wschat.pm
sub accept

```
    ...

    $self->on( message => sub {
        my($ws, $msg) = @_;

        $self->app->log->debug("Got message from ${id}
($client_alias{$id}): ${msg}.");

        return unless $msg && $msg =~ /\S/;

        if ( $id eq $client_alias{$id} and $msg !~ /\b0x\w+\b/ ) {
            $self->app->log->debug("$id is now know as ${msg}.");
            $client_alias{$id} = $msg;
            $msg = "## ${id} is now known as ${msg}.";
        }

        $self->broadcast($msg);
    });

    ...
```

lib/Wstest/Wschat.pm
sub accept

```
    ...

    $self->on( finish => sub {
        delete $client_ws{ $id };
        $self->broadcast("## $client_alias{$id} (${id}) left the
conversation.");
        delete $client_alias{ $id };
        $self->app->log->debug(scalar(keys %client_ws)." Active
clients left.");
    });
}
```

lib/Wstest/Wschat.pm
sub broadcast

```
sub broadcast {
    my( $self, $msg ) = @_;

    my $from_id = $self->tx =~ /(0x\w+)/ && $1;

    if ( $msg =~ /^##(.*)/ ) {
        $self->app->log->debug("${from_id}: $1");
    }

    foreach my $id ( keys %client_ws ) {

        my $prefix =
            $msg =~ /^##/      ? ""    :
            $id eq $from_id    ? "Me" :
                                 $client_alias{$from_id};

        $prefix .= ":  " if $prefix;

        $client_ws{$id}->send( $prefix . $msg );
    }
}
```

Demo

Chat

Examples

- Create new Mojolicious app.

a) Send message, every 3 sec.

b) Chat script – multiple clients talking together.

- **Application testing.**

Test

Test

t/basic.t

```
#!/usr/bin/perl

use strict;
use warnings;

use Test::More tests => 42;
use Test::Mojo;
use FindBin qw($Bin);
use lib "$Bin/../lib";

my $t = Test::Mojo->new('Wstest');

...
```

Test

t/basic.t

```
...

#
#=== Test Timer part
#

$t ->get_ok('/')
   ->status_is(200)
   ->content_like(qr/Demo Page/i);

my $tt = $t->websocket_ok('/time')
   ->send_ok('hello', 'Sent messages are ignored');

diag("Expexct to wait 3 secs for timed message to be sent");

$tt->message_ok
   ->message_like(qr/\b\d{4}\b/)  # Match year of timestamp.
   ->finish_ok;

...
```

Test

t/basic.t

```
...

#
#=== Test Chat part
#

my $c1 = $t->websocket_ok('/chat');

$c1->message_ok->message_like(qr/Nobody else to talk to at the
moment/);
$c1->message_ok->message_like(qr/You are known as (0x\w+)/);
my($id1) = map { /(0x\w+)/ ? $1 : $_ } $t->message->[1];
#diag("Test id1 $id1");
$c1->message_ok->message_like(qr/Please send your name as the first
message/);


...
```

Test

t/basic.t

```
...

my $c2 = $t->websocket_ok('/chat');

$c1->message_ok->message_like(qr/joined the conversation/);

$c2->message_ok->message_like(qr/Talking to: $id1/);
$c2->message_ok->message_like(qr/You are known as 0x\w+/);
my($id2) = map { /(0x\w+)/ ? $1 : $_ } $t->message->[1];
#diag("Test id2 $id2");
$c2->message_ok->message_like(qr/Please send your name as the first
message/);

...
```

Test

t/basic.t

```
...

$c1->send_ok('C1', 'Sent message from c1');
$c1->message_ok->message_like(qr/\Q${id2}\E is now known as C1/);
$c2->send_ok('C2', 'Sent message from c2');
$c2->message_ok->message_like(qr/\Q${id2}\E is now known as C1/);

$c1->message_ok->message_like(qr/C1:  C2/);
$c2->message_ok->message_like(qr/Me:  C2/);

#$c1->send_ok('hello 1 from c1', 'Sent message from c1');
#$c2->message_ok->message_like(qr/C1:  hello 1 from c1/);
#$c1->message_ok->message_like(qr/Me:  hello 1 from c1/);

#$c2->send_ok('hello 2 from c2', 'Sent message from c2');
#$c1->message_ok->message_like(qr/C1:  hello 2 from c2/);
#$c2->message_ok->message_like(qr/Me:  hello 2 from c2/);

$c1->finish_ok;
$c2->finish_ok;

# done
```

Test

t/basic.t

```
$ prove
t/basic.t .. 1/36 # Expexct to wait 3 secs for timed message to be
sent
t/basic.t .. ok
All tests successful.
Files=1, Tests=36,  4 wallclock secs ( 0.05 usr  0.01 sys +  0.42
cusr  0.03 csys =  0.51 CPU)
Result: PASS
```

Live demo

Live demo

Now you can chat

`http://127.0.0.1:3000/chat/`

Downloads

`http://www.uniejo.dk/presentations/2014-05-24.pdf`
This presentation

`http://www.uniejo.dk/presentations/2014-05-24.tgz`
Gzipped tar archive of files used in this presentation.

Files

```
$ find . -type f
./lib/Wstest.pm
./lib/Wstest/Wschat.pm
./lib/Wstest/Wstime.pm
./lib/Wstest/Wswork.pm
./log/development.log
./public/css/mystyle.css
./public/js/jquery.js
./public/js/wschat.js
./public/js/wstime.js
./script/wstest
./t/basic.t
./templates/layouts/default.html.ep
./templates/wschat/welcome.html.ep
./templates/wstime/welcome.html.ep
./templates/wswork/welcome.html.ep
./wstest.conf
```

End of presentation...