Node.js - Inheritance

- We have already seen prototypal inheritance
- However, Node includes its own convenience mechanism for inheritance
- For future reference: It is similar to `_.extend`, but Node does not depend on `_` (Underscore)

```
1.  'use strict';
2.  const
3.    events = require('events'),
4.    util = require('util'),
5.    // event client constructor
6.    EventsClient = function() {
7.      events.EventEmitter.call(this);
8.    };
9.  util.inherits(EventsClient, events.EventEmitter);
10.
11. const client = new EventsClient();
```

```
12. client.on("myEvent", function() {
13.   console.log("myEvent triggered");
14. })
15. client.emit("myEvent");
```

example*code/inheritance/inheritance*with_node.js

- Calling the `EventEmitter` constructor in the EventsClient constructor is roughly equivalent to calling `super()` in classical OOP
- Calling `util.inherits()` makes EventsClients' prototypal parent object the EventEmitter prototype

Node.js - Modules

- One of the biggest problems in JS are naming collisions
- Say you have included a datepicker and a day-by-day schedule
  - *Both could define a* `drawCalendar()` *function*
  - *If the names are declared in the global namespace at least one program breaks*
- Remember: This is why jQuery has its `$` namespace
- Node has a generic solution to this problem called *Modules*

Node.js - Modules

- Node modules follow the *Module Design Pattern*
- The pattern was originally defined as a way to provide both private and public encapsulation for classes
- In JS, it is used to emulate the concept of classes - providing public/private methods and variables inside a single object
- Read more about the Module Design Pattern in Essential JS Design Patterns by Addy Osmani

Node.js - Modules

- A module is just a JS file
- However, this file is evaluated in a special way
- When Node loads the file, it creates a new scope, so the previous datepicker could not mess with the scheduler plugin
- From inside the plugin, the outside world is invisible
  - *Except for requiring other modules*
- As a result, your code is properly scoped - no collisions possible

Node.js - Modules

- Exposing *public* methods or variables is done by scoping them under `exports`

Example module

```
1. f1 = function() {
2.   console.log("this is a private function and cannot be called outside
   this module");
3. }
4.
5. // public_f1 exposes the otherwise private function f1
6. exports.public_f1 = f1;
7.
```

Example application

```
1. var sample = require('./sample_module');
2.
3. // this call would fail with: TypeError: Object #<Object> has no method
```

```
'f1'
// sample.f1();

// this call will work
sample.public_f1();
```

Node.js - Modules

- Using parts of modules is done similarly
- But saving only a reference to a member of the returned exports object

```
1. var public_f1 = require("./sample_module").public_f1;
2.
3. public_f1();
```

- This might save you some typing

Faye - Robust Messaging Services

- We're going to explore how to write robust messaging services in Node
- We are saving you the low-level capabilities of Node
  - *If interested, read up on Node and sockets in Chapter 3 of NTRW (see resources)*
- As messaging service, the cross-platform library Faye will be used
- We have heard about Faye in Web2 when discussing Websockets
- Faye is a publish-subscribe messaging system based on the Bayeux protocol
  - *It provides*
    - Message servers for Node.js and Ruby
    - And clients for use on the server and in all major web browsers

Faye - PubSub Pattern

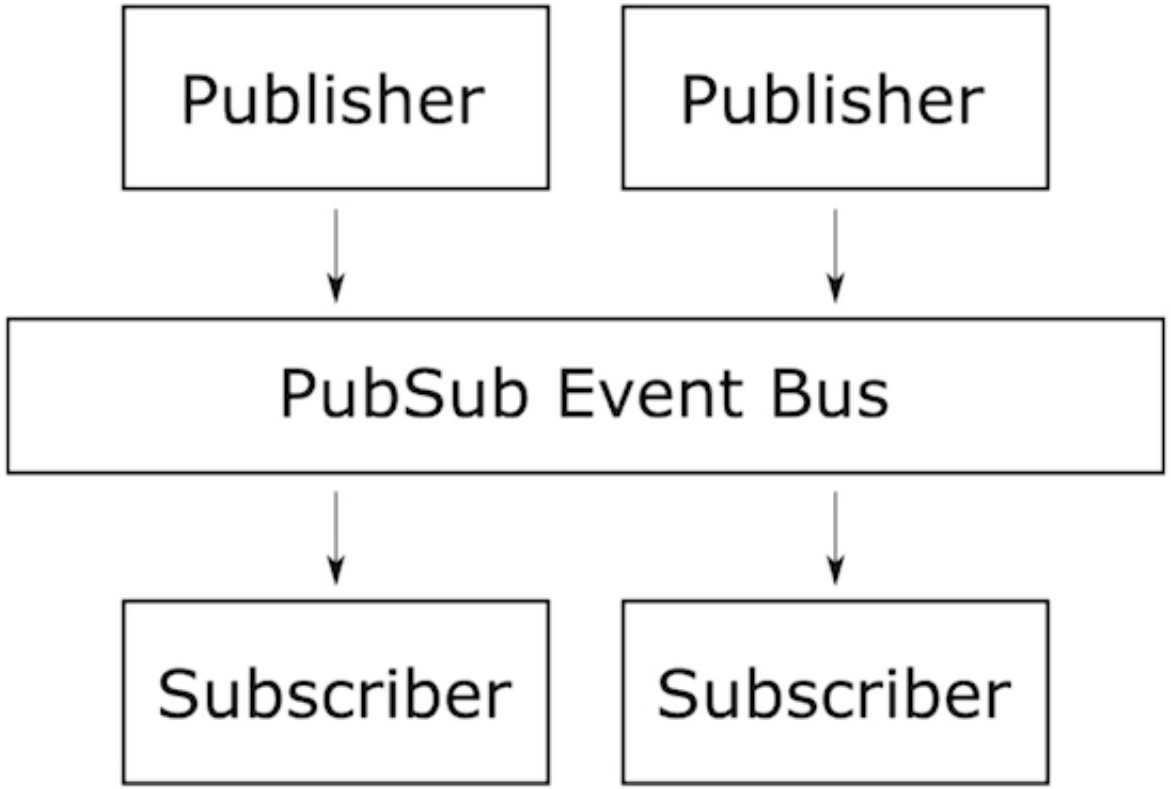
- PubSub is a messaging pattern where senders of messages are called *publishers*
- These messages are not programmed to be sent directly to receivers, called *subscribers*
- Instead, published messages are characterized into classes, without knowledge of what, if any, subscribers there may be
- Similarly, subscribers express interest in one or more classes
- They only receive messages that are of interest, without knowledge of what, if any, publishers are there

Faye - PubSub Pattern

- The largest benefit of using pub/sub is the ability to break down the app into smaller, more loosely coupled modules
- It encourages to think about relationships between parts of the app
  - *Identifying what layers need to observe or listen for behaviour*
  - *And which need to push notifications regarding behaviour occurring in other parts of our apps*
- It is good for designing decoupled systems and are an important tool in the JS toolbelt

Faye - PubSub Pattern

- PubSub is structured as an Event Bus as follows



Faye - First example

- Faye is not your usual messaging system in that it provides backend *and* frontend clients
- Making it very easy to use PubSub
- Also it takes good care of as a transport layer
- It uses the best mechanism available to your browser, in the order of
    1. *Websockets*
    2. *Long-Polling via HTTP Post (XHR)*
    3. *Cross Origin Resource Sharing (*CORS*)*
    4. *Callback-polling (*JSONP*)*

Faye - First example

Server Side

1. Install Faye: `npm install faye`
2. Start a server

```
1. var http = require('http'),
2.     faye = require('faye');
3.
4. var server = http.createServer(),
5.     bayeux = new faye.NodeAdapter({mount: '/'});
6.
7. bayeux.attach(server);
8. server.listen(8000);
```

`example_code/faye/first_example/backend/server.js`

Faye - First example

Client Side

- Subscribing to a channel is easy

```
1. var client = new Faye.Client('http://localhost:8000/');
2.
3. client.subscribe('/messages', function(message) {
4.   alert('Got a message: ' + message.text);
5. });
```

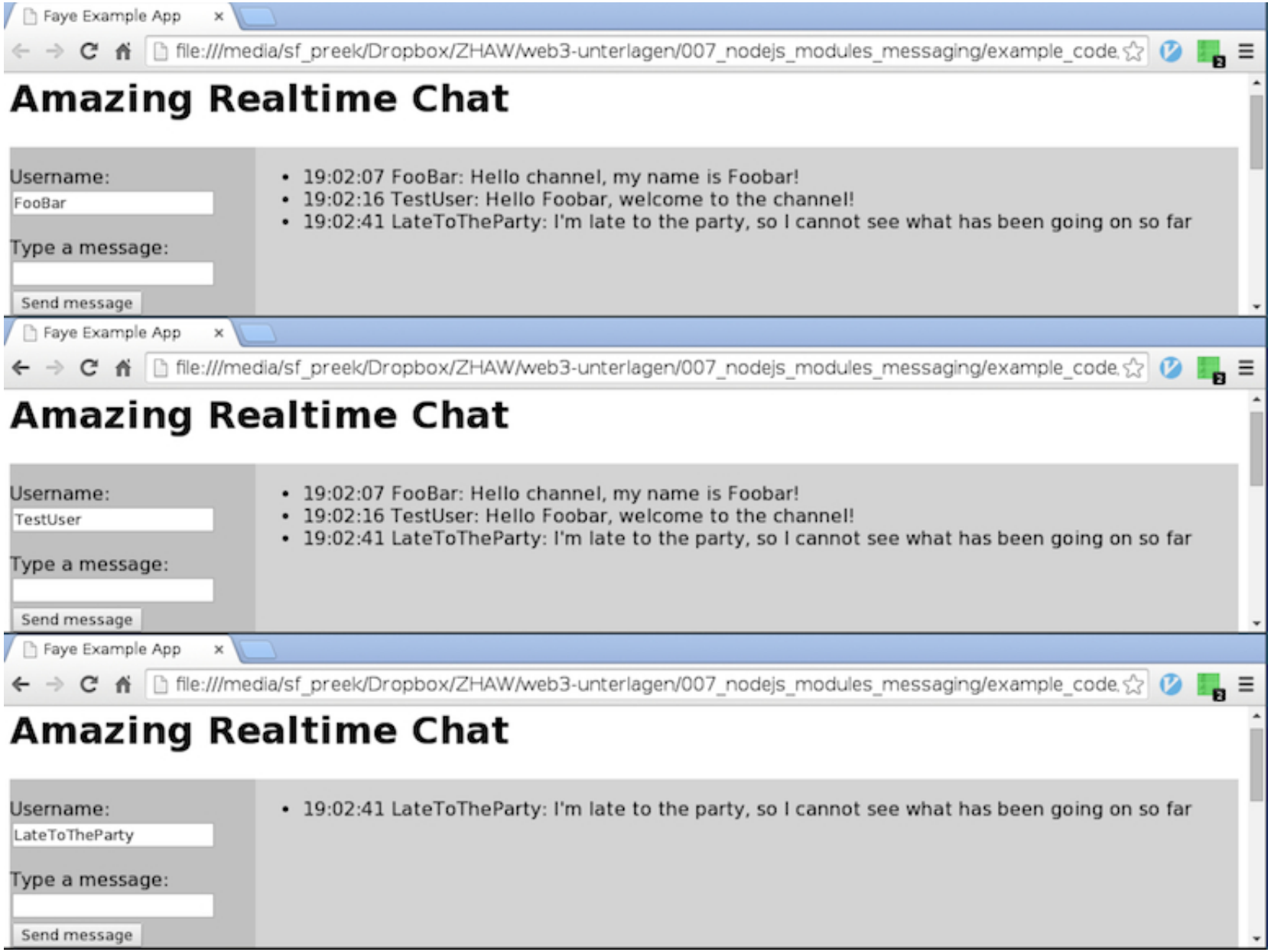
- Notice how nobody is sending to "/messages" as of now
- Whence somebody is sending, however, all is in place already

```
1. client.publish('/messages', {
2.   text: 'Hello world'
3. });
```

Try the example code in: `example_code/faye/first_example/frontend`

Faye - Chat Application

- This is an example running the very same server as above
- There's just a tiny bit of CSS and jQuery going on to be able to send different messages


Faye Example App
file:///media/sf_preek/Dropbox/ZHAW/web3-unterlagen/007_nodejs_modules_messaging/example_code
Amazing Realtime Chat
Username:
FooBar
Type a message:
Send message
19:02:07 FooBar: Hello channel, my name is Foobar!
19:02:16 TestUser: Hello Foobar, welcome to the channel!
19:02:41 LateToTheParty: I'm late to the party, so I cannot see what has been going on so far
Faye Example App
file:///media/sf_preek/Dropbox/ZHAW/web3-unterlagen/007_nodejs_modules_messaging/example_code
Amazing Realtime Chat
Username:
TestUser
Type a message:
Send message
19:02:07 FooBar: Hello channel, my name is Foobar!
19:02:16 TestUser: Hello Foobar, welcome to the channel!
19:02:41 LateToTheParty: I'm late to the party, so I cannot see what has been going on so far
Faye Example App
file:///media/sf_preek/Dropbox/ZHAW/web3-unterlagen/007_nodejs_modules_messaging/example_code
Amazing Realtime Chat
Username:
LateToTheParty
Type a message:
Send message
19:02:41 LateToTheParty: I'm late to the party, so I cannot see what has been going on so far

Faye - Chat Application

- Notice that there are three browsers open
- Each has registered a different username
- The two upper browser windows seem to have been open first (see chat text)
  - *So they were able to send and retrieve messages*
- The browser on the bottom opened the app later
  - *However, when the third user writes, all existing users can read the message*
- Hence, in very little code, we have a 'complete' asynchronous chat client and server

Try the example code in: *example_code/faye/chat*

Resources

- Node.js the Right Way: Practical, Server-Side JavaScript That Scales
- Creating Custom Modules
- Module Pattern in Javascript
- Understanding the Publish/Subscribe Pattern for Greater Javascript Scalability
- CORS
- JSONP