How to build Large Web apps

Peter Svensson, Greener Grass Web Design AB
psvensson@gmail.com

What do I do?

greener grass

GTUG Stockholm

What you need to do

1. Managing JavaScript files
2. Using OO to DRY
3. Client-side templates
4. Intra-App communication strategies
5. Examples

JavaScript File Management

Problem (Stolen from requirejs.org)

- *Web sites are turning into Web apps*
- *Code complexity grows as the site gets bigger*
- *Assembly gets harder*
- *Developer wants discrete JS files/modules*
- *Deployment wants optimized code in just one or a few HTTP calls*

JavaScript File Management (contd.)

Front-end developers need a solution with:

- *Some sort of #include/import/require*
- *ability to load nested dependencies*
- *ease of use for developer but then backed by an optimization tool that helps deployment*

AMD - Asynchronous Module Definition

- Requirejs (and Dojo, et.c.) implements AMD
- AMD is a common module format for JS
- A module exists in a namespace (foo.bar.myclass)
- A module can refer to other modules

https://github.com/amdjs/amdjs-api/wiki/AMD
http://requirejs.org/docs/whyamd.html

What does it look like then?

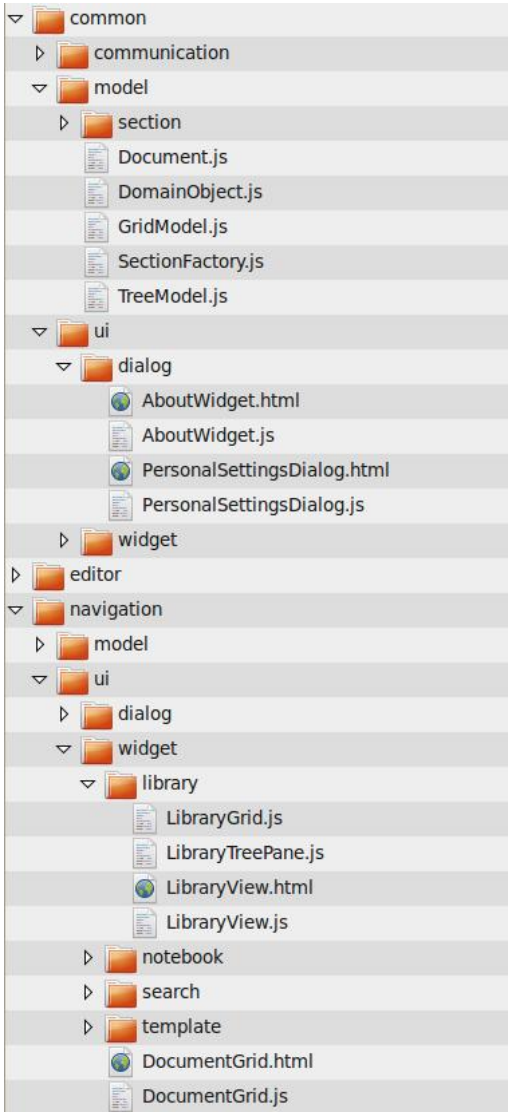
1)

```
define(['jquery'] , function ($) {
    return function () {};
});
```

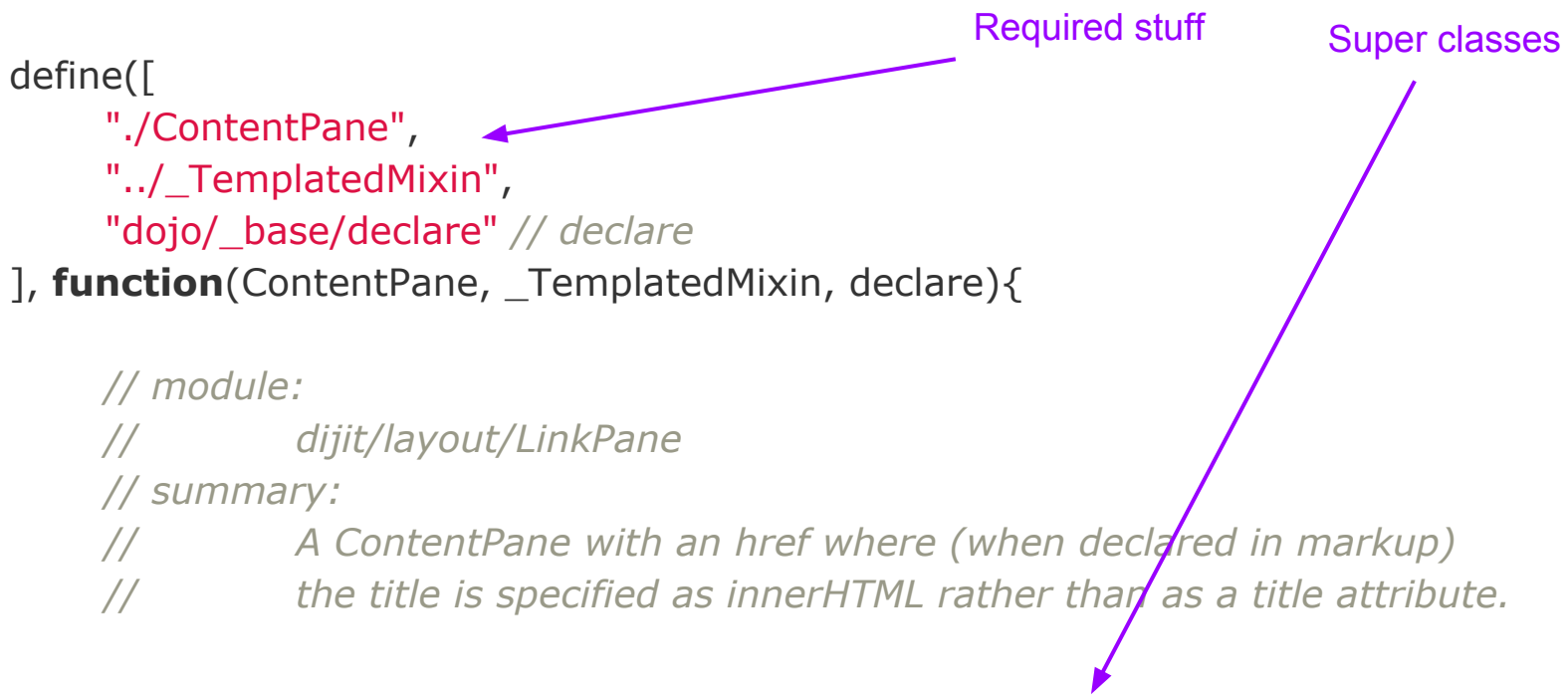
2)

```
define(["../_base/lang","../_base/declare","../_base/fx", "../_base/connect"],
  function(lang, declare, baseFx, connectUtil) {

    return declare("dojo.fx.Toggler", null, {
        .....
    });
});
```

What the file structure looks like



Using OO to DRY

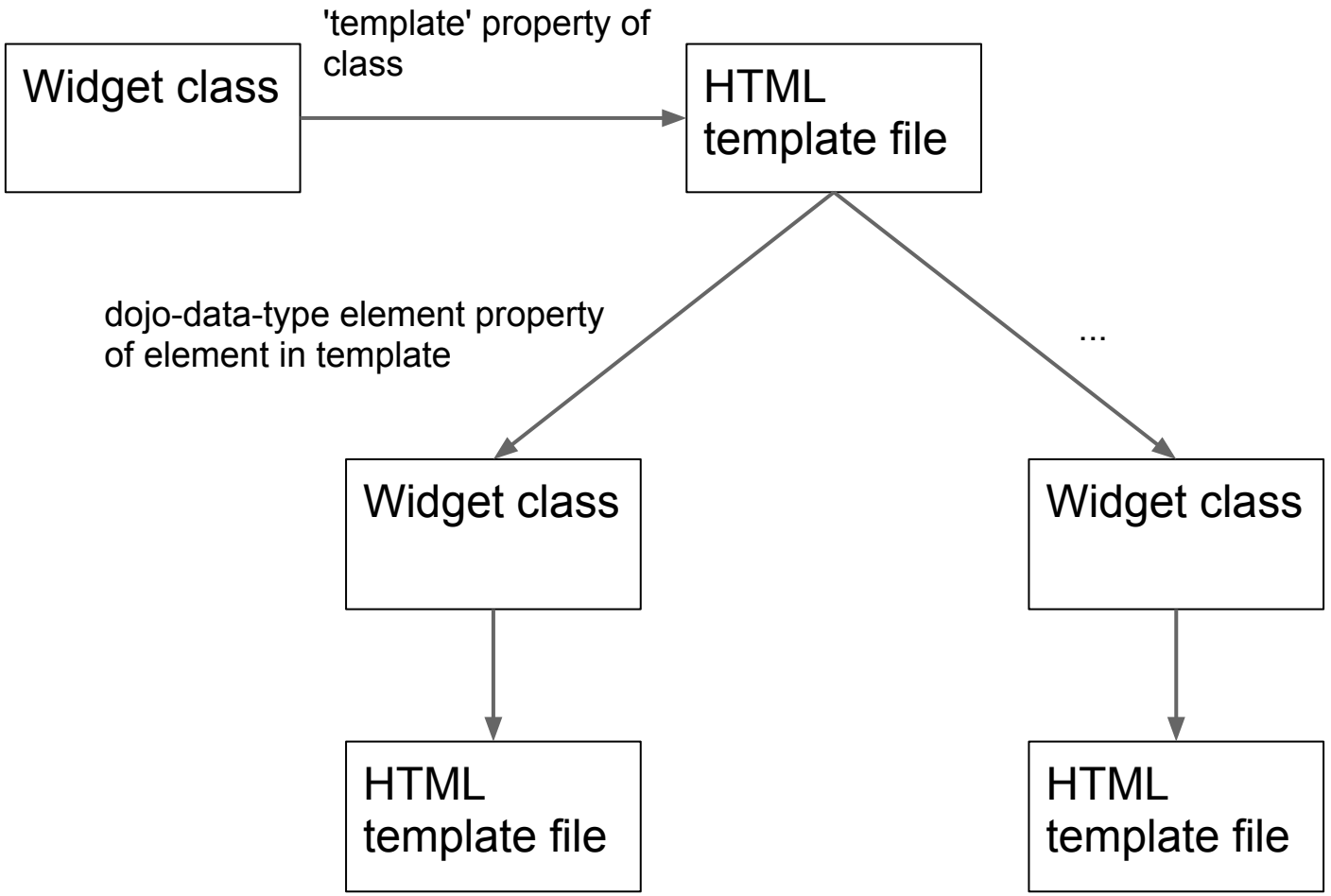



```
define([
        "./ContentPane",
        "../_TemplatedMixin",
        "dojo/_base/declare" // declare
], function(ContentPane, _TemplatedMixin, declare){

        // module:
        //          dijit/layout/LinkPane
        // summary:
        //          A ContentPane with an href where (when declared in markup)
        //          the title is specified as innerHTML rather than as a title attribute.


        return declare("dijit.layout.LinkPane", [ContentPane, _TemplatedMixin], {
            ......
            }
    })
);
```

So what was AMD good for again?

- An asynchronous loader of your JS files
- A namespaced module format
- A way to implement OO/Multiple inheritance
- Forces you to do the right thing
- Simple wrappers exist for most JS libs
- If you don't use it, you're *doomed* to reimplement parts of it, badly (again :)

Client-side templates (Dojo example)

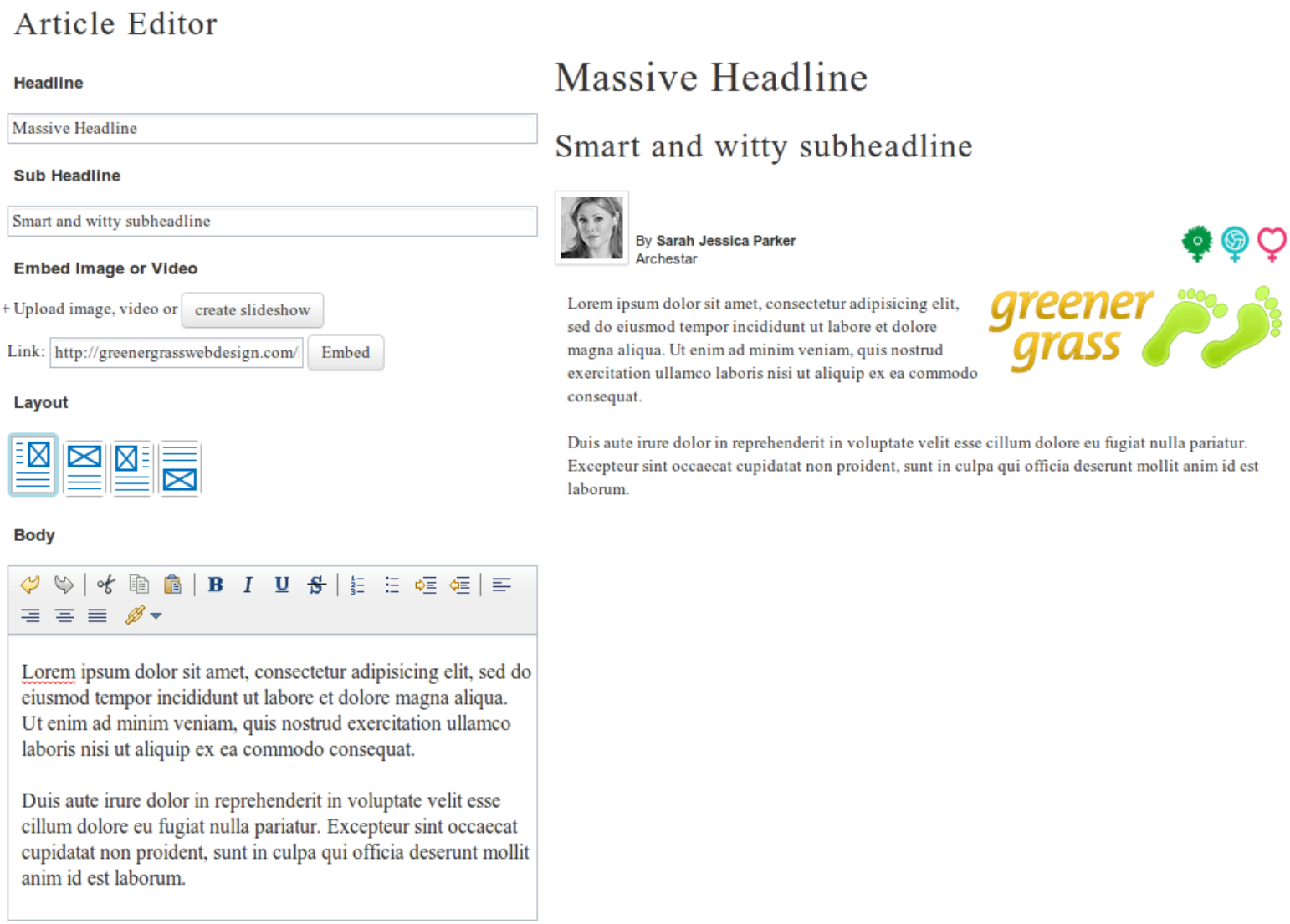


Client-side templates

```
define([
        "dojo/_base/array", // array.forEach
        "dojo/_base/declare", // declare
        ...
        "dojo/text!./templates/HorizontalSlider.html"
], function(array, declare, ..., template){
....
    var HorizontalSlider = declare("dijit.form.HorizontalSlider", [_FormValueWidget, _Container], {

                templateString: template,
                foo: 17,
                doSomething: function(){ this.sliderHandle.innerHTML = "foobar"; }
....
    });
return HorizontalSlider;
});

-------------------------------- Part of template file
..
<div class="dijitSliderMoveable dijitSliderMoveableH">
  <div data-dojo-attach-point="sliderHandle,focusNode" data-dojo-attach-event="press:
_onHandleClick" role="slider"${foo}></div>
</div
..
```

Template

MOAR client-side templates



What the index.html looks like

```
<body>
    <div class="claro">
        <h2>Article Editor</h2>
        <div data-dojo-type="cms.editor" ... >
        </div>
    </div>
</body>
```

Why client-side templates?

Separating **looks** and **logic** in widgets.

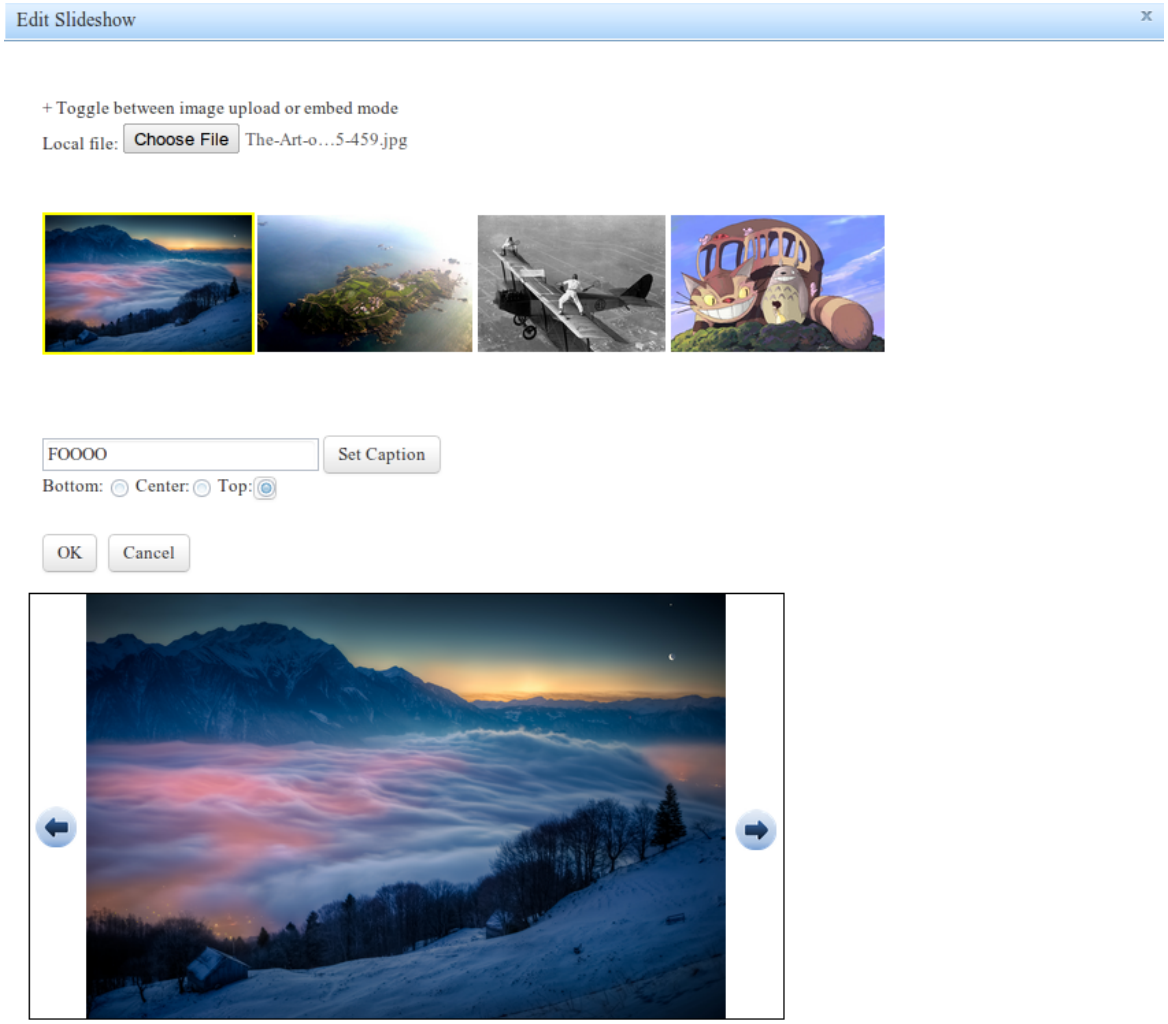
Easy-peasy conversion of design->widget.

No need for id/class lookup, using **symbolic names** for DOM elements.

Means you can at no cost crank out two or a hundred of a widget knowing it will not collide.

Handlebars up class properties into template before rendering (e.g. ${username})

Even MOAR .. OK, you get it.



Intra-App communication strategies

1. Why do you need to do that?
2. The object-passing approach
3. The global object approach
4. The pubsub approach

Conceptual Distance - Example

mainWidget creates **editWidget** and **navigationWidget**

editWidget creates **statusWidget**

navigationWidget create **loginWidget**

statusWidget needs to know when **loginWidget** have logged in or out a user

Object-passing to cover distance

mainWidget passes a reference of itself to the widgets it creates

Each widget continue to pass the **mainWidget** reference to their widgets, and so on.

When any widget need to communicate with another, they use specific functions on the **mainWidget**.

Global object to cover distance

For example;

```
window.foo = {};
window.foo.bar = 4711;
window.foo.baz = function(args){ ... }
```

All object anywhere can now access window.foo.bar and baz();

(Works because of the single-threaded browser model)

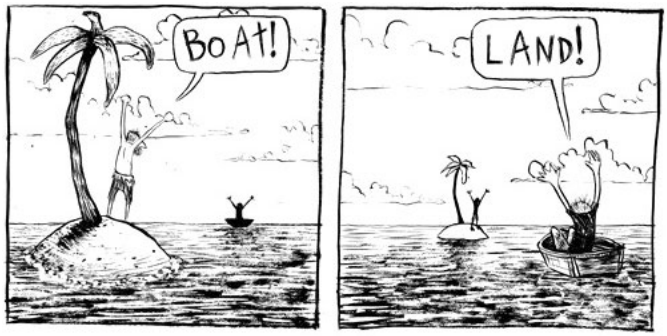
Using pubsub to cover distance

In statusWidget:

```
dojo.subscribe("channelname", function(arg)
{
    console.log("we got "+arg+" from channel");
});
```

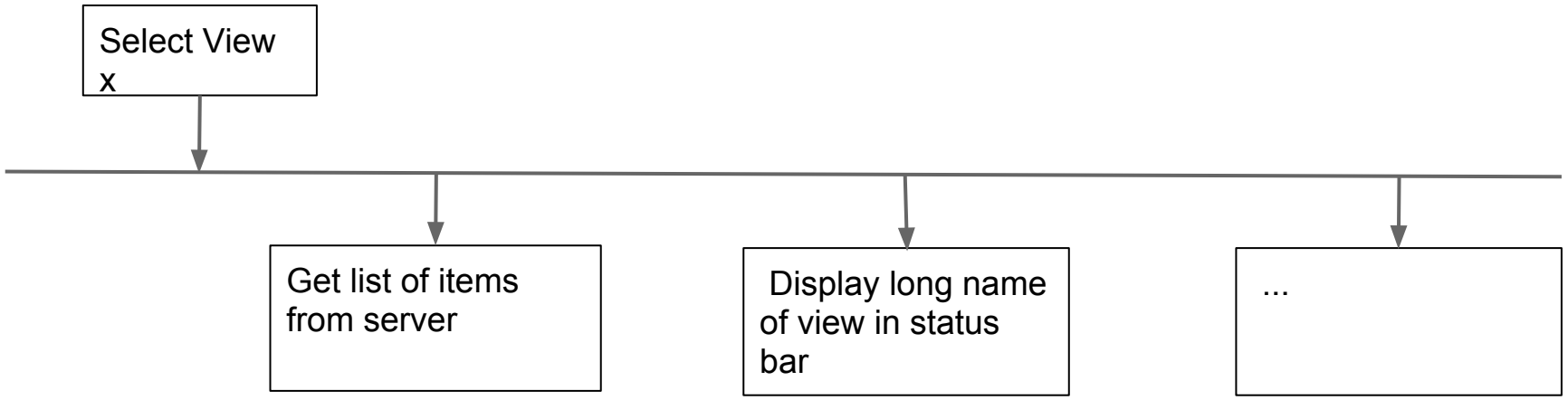
In loginWidget:

```
dojo.publish("channelname", [17] );
```

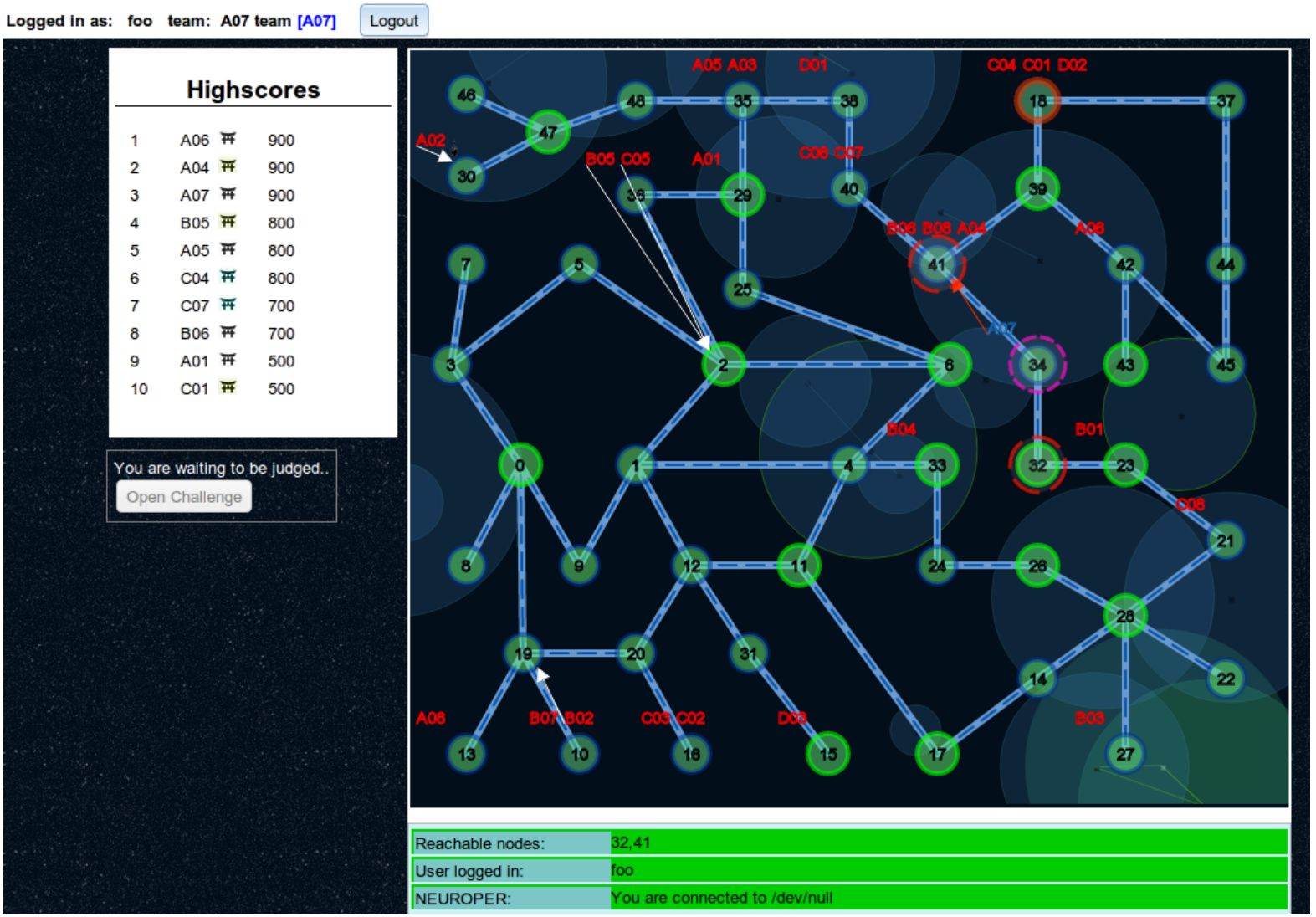


Best practices

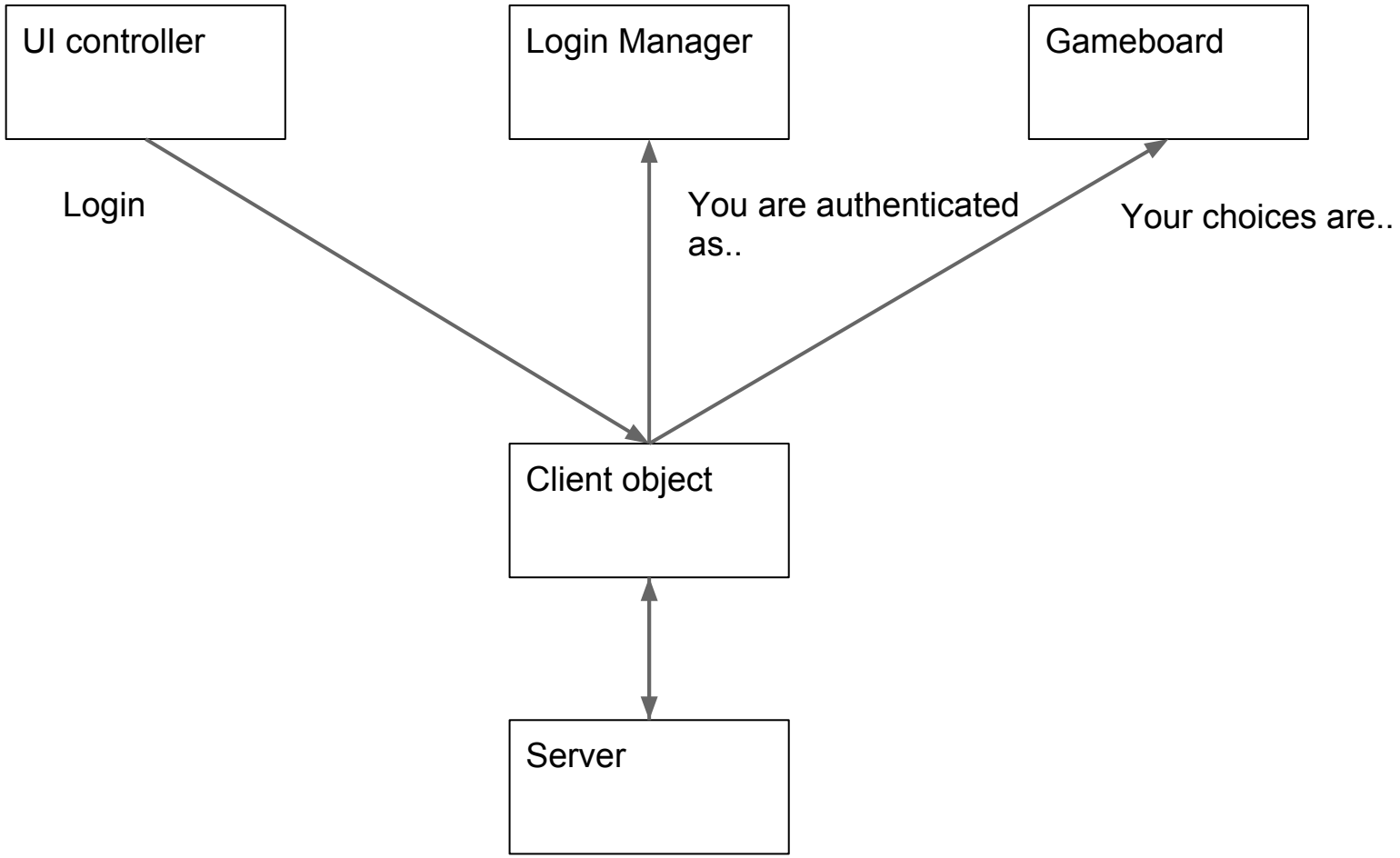
1. Use a global object
2. Put all server-access, all centralized error handling and all common data massage there.
3. Use Pubsub messaging for EVERYTHING



Event-driven Example (from swdc.se/devnull)



Event-driven example



Questions?