JavaScript APIs

Modern APIs that will make your apps more userfriendly

Agenda

History API

Instrument for processing and creating files in browser

getUserMedia

“Link” to binary file representation in memory

Geolocation API

Retrieve location of user’s device

Fullscreen API

Allows application to be shown in full-screen mode

postMessage

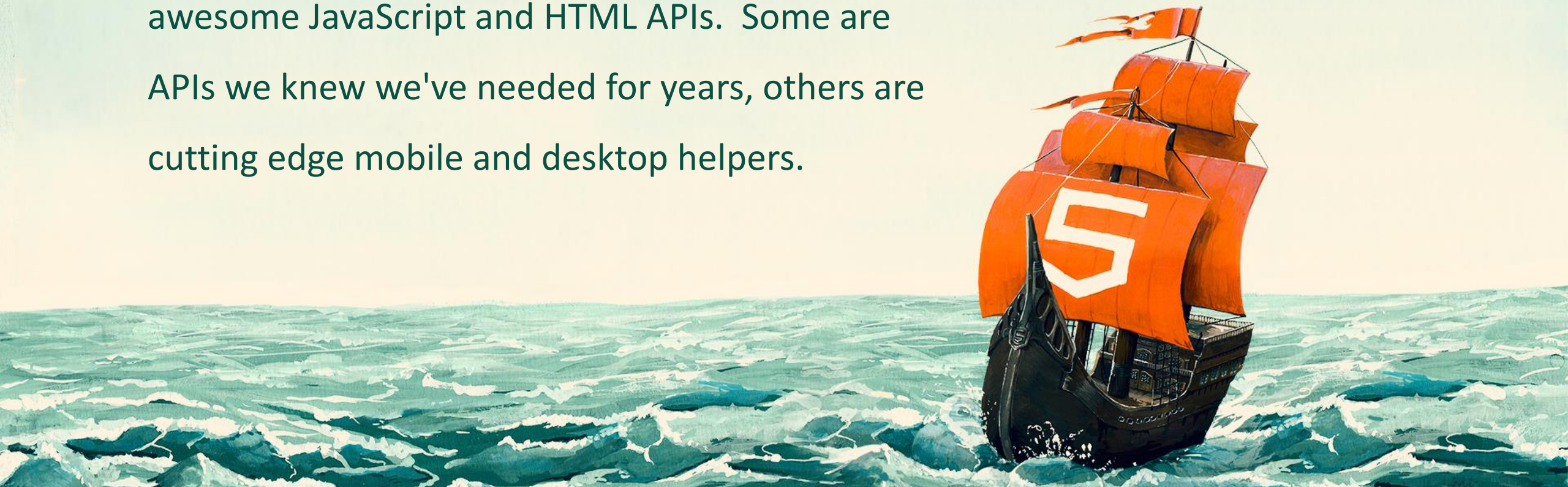
Secure data exchange, sandboxing

Workers API

Run scripts in background threads

Introduction

The HTML5 revolution has provided us some awesome JavaScript and HTML APIs. Some are APIs we knew we've needed for years, others are cutting edge mobile and desktop helpers.




HTML Markup
Audio/Video
Canvas
2D Drawing
Microdata
Web Workers
HTML5
Web Storage
Communication
Timed media playback
Browser history management
Offline Web Applications
Document Editing
Drag and Drop

History API

Control address bar and application navigation


MARTY WE CAN CONTROL
BROWSER HISTORY

History API

Motivation

Proper URL in address bar

Friendly for search engines

Easy handling of Back button

Easy communication with server

Store website/application state

Hash and hashbang are not sent to the server

History API

window.history interface

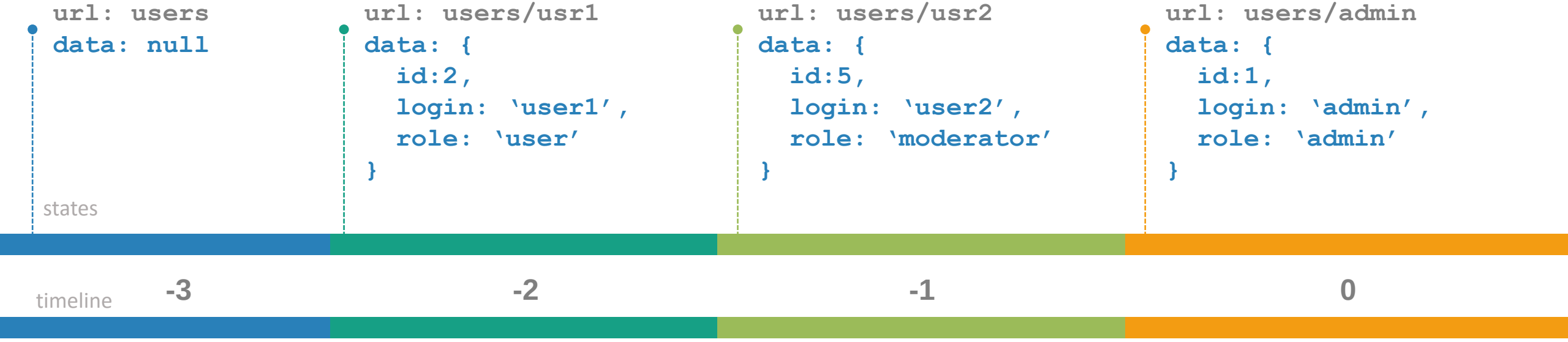
JAVASCRIPT

```
window.history.state //current state object
window.history.length //history length
window.history.go(delta)//goes back/forward for delta steps
window.history.back()//same as go(-1)
window.history.forward()//same as go(1)
window.history.pushState(data, title, url)
window.history.replaceState(data, title, url)
```

History API

States. Initial

- You can imagine history as line of states.



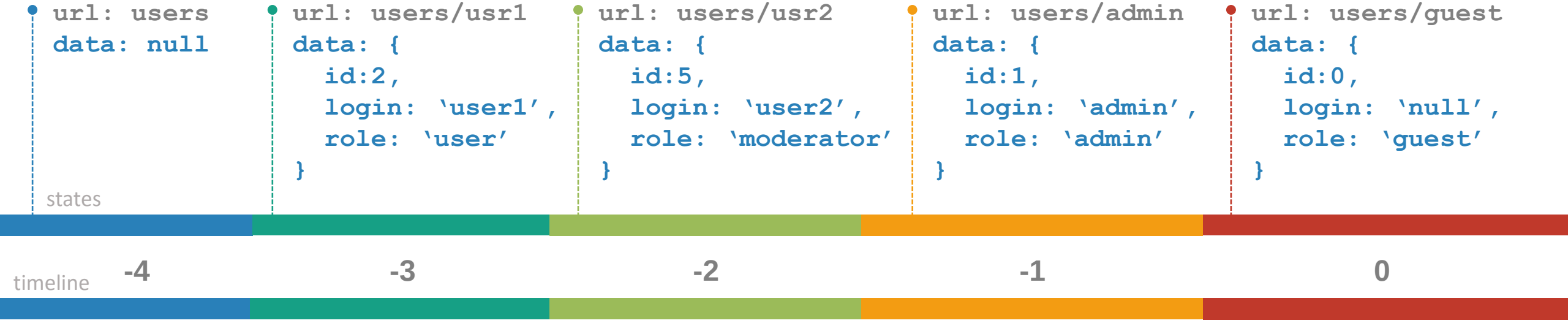
History API

States. pushState

JAVASCRIPT

```
var guest = {id: 0, login: null, role: 'guest'};

window.history.pushState(guest, '', 'users/guest');
```

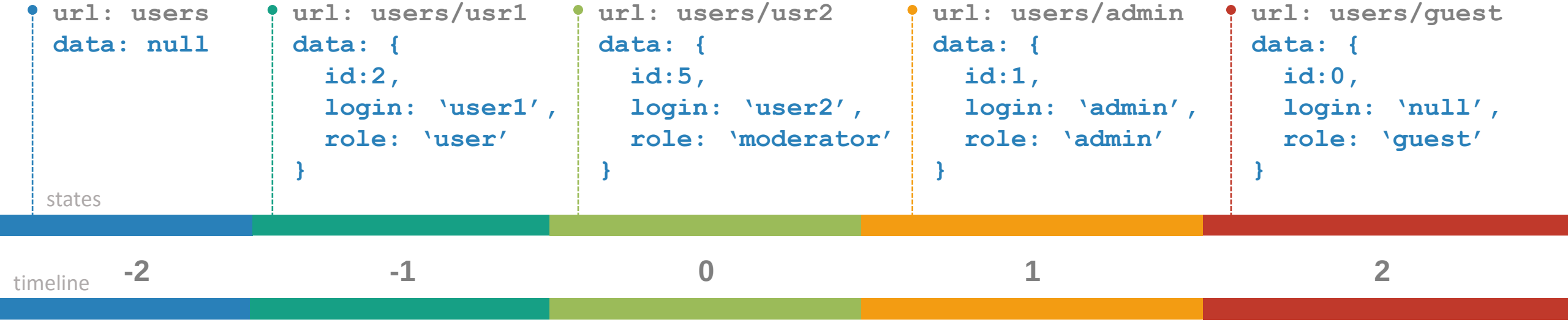


History API

States. go back

JAVASCRIPT

```
window.history.go(-2);
```

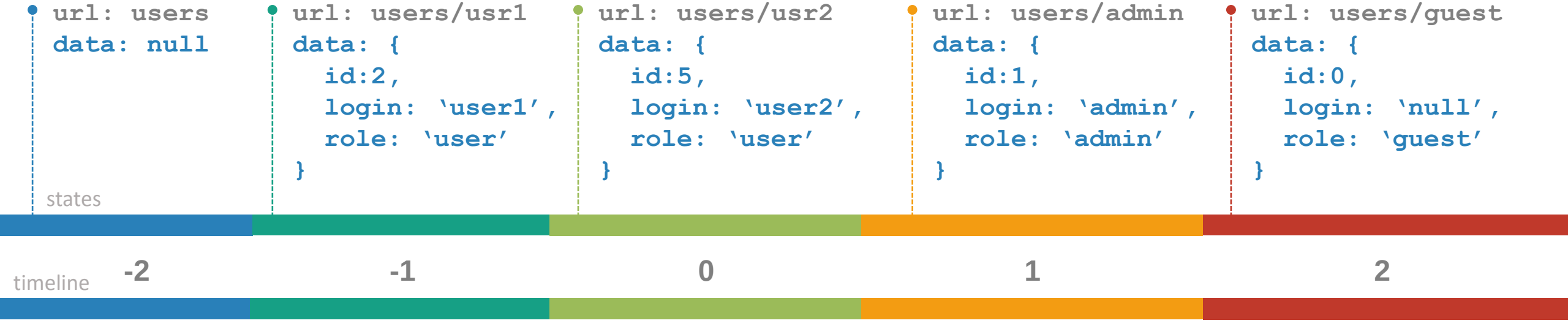


History API

States. replaceState

JAVASCRIPT

```
window.history.replaceState({id: 5, login: 'user2', role: 'user'}, '');
```



History API

Track state change

JAVASCRIPT

```
window.onpopstate = function (event) {
    console.log(window.location); // http://localhost:3000/users/user2
    console.log(event.state); // http://localhost:3000/users/user2
};
```

- Don’t forger to change **`document.title`** ;)

History API

Routing basics

Routing – process of mapping of URL (or hash) to the function(s) that will render required result.

History API

Routing options

Hash based

WHEN TO PERFORM ROUTING

- DOMContetnLoaded or load event on window object
- hashchange event on window object

History API based

WHEN TO PERFORM ROUTING

- DOMContetnLoaded or load event on window object
- Click on any link for this host
- popstate event on window object

History API

Track current URL

To get access to the current page URL you can use window.location of document.location objects which have list of properties which correspond to different parts of the URL. The same set of properties also available on anchor tags (<a>)

History API

Track current URL

History API based approach uses:

- Parsing current page URL - `window.location.pathname`
  (`window.location.pathname` + `window.location.search` in rare cases)
- Determine if link is for current web page - `linkElement.host` and `window.location.host`
- Parsing currently clicked link path - `linkElement.pathname`
- Also special methods of your application (e.g. `app.navigate`) may be used to programmatically decide and navigate to URL after users or server actions

getUserMedia

Get access to video and audio inputs from browser

getUserMedia

Part of Media Capture and Streams – API that allow local media, including audio and video, to be requested from a platform.

Method of `navigator` object – asks permission to access video and audio streams.

getUserMedia

Request access

JAVASCRIPT

```
navigator.getUserMedia = navigator.getUserMedia
                       || navigator.webkitGetUserMedia
                       || navigator.mozGetUserMedia
                       || navigator.msGetUserMedia;

if (navigator.getUserMedia) {
    navigator.getUserMedia(constraints, successCb, errorCb);
} else {
    alert('No fun with your browser');

}
```

getUserMedia

Get and use stream

JAVASCRIPT

```
function successCb(stream) {
    video.src = URL.createObjectURL(stream);
    //if needed for later use
    mediaStream = stream;
}

function errorCb() {
    alert('Error: ' + error);
}
```

- Demo: http://jsfiddle.net/mnsnroy2/

Geolocation API

Retrieve location of user's device

Geolocation API

Motivation

Allows to obtain user location

NAVIGATION

LOCATION SPECIFIC ACTION

LOCATION SPECIFIC ADS/NOTIFICATIONS

Geolocation API

navigator.geolocation interface

JAVASCRIPT

```
navigator.geolocation.getCurrentPosition(success, error, options);

navigator.geolocation.watchPosition(success, error, options);
//returns id of watcher (watcherId)

navigator.geolocation.clearWatch(watcherId);
```

Available options:

- `enableHighAccuracy` – notifies browser to use best available position provider
- `timeout` – how long we wait for results
- `maximumAge` – how long we want to use cached value

Geolocation API

Success callback. Position object

- **`timestamp`** – when Position object was acquired
- **`coords`** – Coordinates object:
  - `latitude`
  - `longitude`
  - `altitude`
  - `accuracy`
  - `altitudeAccuracy`
  - `Heading` – 0 (north), 90 (east), 180 (south), 270 (west)
  - `speed`

Geolocation API

Error callback. PositionError object

- `code` – error code. One of:

```
PositionError.PERMISSION_DENIED //(1)
PositionError.POSITION_UNAVAILABLE //(2)
PositionError.TIMEOUT //(3)
```

- `message` – error message

Geolocation API

Example

JAVASCRIPT

```
navigator.geolocation.getCurrentPosition(
    function (position) {
        for (var coord in position.coords) {
            document.body.innerHTML +=
                '<p>' + coord + ': '
                + position.coords[coord] + '</p>';
        }
    }, function (error) {
        alert('Error ' + error.code + '\n' + error.message);
    }, {
        enableHighAccuracy: true
    });
```

Geolocation API

Example. Watcher

JAVASCRIPT

```
var startWatcher = document.getElementById('start');
var stopWatcher =document.getElementById('stop');

startWatcher.onclick = function() {
    watcher = navigator.geolocation
        .watchPosition(successCb, errorCb, options);
};

stopWatcher.onclick = function() {
    navigator.geolocation.clearWatch(watcher);
};
```

Geolocation API

Demo

http://jsfiddle.net/kv1dsdLc/embedded/result,js/

Fullscreen API

Allows application to be shown in full-screen mode

Fulscreen API

Fullscreen interface

Allows to switch whole document or element to full screen mode programmatically and apply special styling.

Element / Document:

```
requestFullscreen()
```

Note: this method should be called *only from event* handler

Document only:

```
document.exitFullscreen()
document.fullscreenEnabled
document.fullscreenElement
document.onfullscreenchange
document.onfullscreenerror
```

Fulscreen API

Cross-browser differences

- `requestFullscreen`
  - webkitRequestFullscreen
  - msRequestFullscreen
  - mozRequestFull**S**creen
- `exitFullscreen`
  - webkitExitFullscreen
  - msExitFullscreen
  - moz**Cancel**Full**S**creen
- `fullscreenElement`
  - webkitFullscreenElement
  - msFullscreenElement
  - mozFull**S**creenElement
- `fullscreenEnabled`
  - webkitFullscreenEnabled
  - msFullscreenEnabled
  - mozFull**S**creenEnabled

Fulscreen API

Cross-browser differences

- `onfullscreenchange`

  `onwebkitfullscreenchange`

  `onmsfullscreenchange`

  `onmozfullscreenchange`

- `onfullscreenerror`

  `onwebkitfullscreenerror`

  `onmsfullscreenerror`

  `onmozfullscreenerror`

Fullscreen API

Styles

Element on which request fullscreen was called:

- :-webkit-full-screen
- :-moz-full-screen
- :-ms-fullscreen
- :full-screen
- :fullscreen

Backdrop:

- ::-ms-backdrop
- ::backdrop

You can have *different presentation and functionality* for normal and fullscreen states of same element *by styling nested elements*.

Fullscreen API

Demo

Check Fullscreen_example in materials folder

postMessage

Secure data exchange

postMessage & cross-origin messaging

Motivation

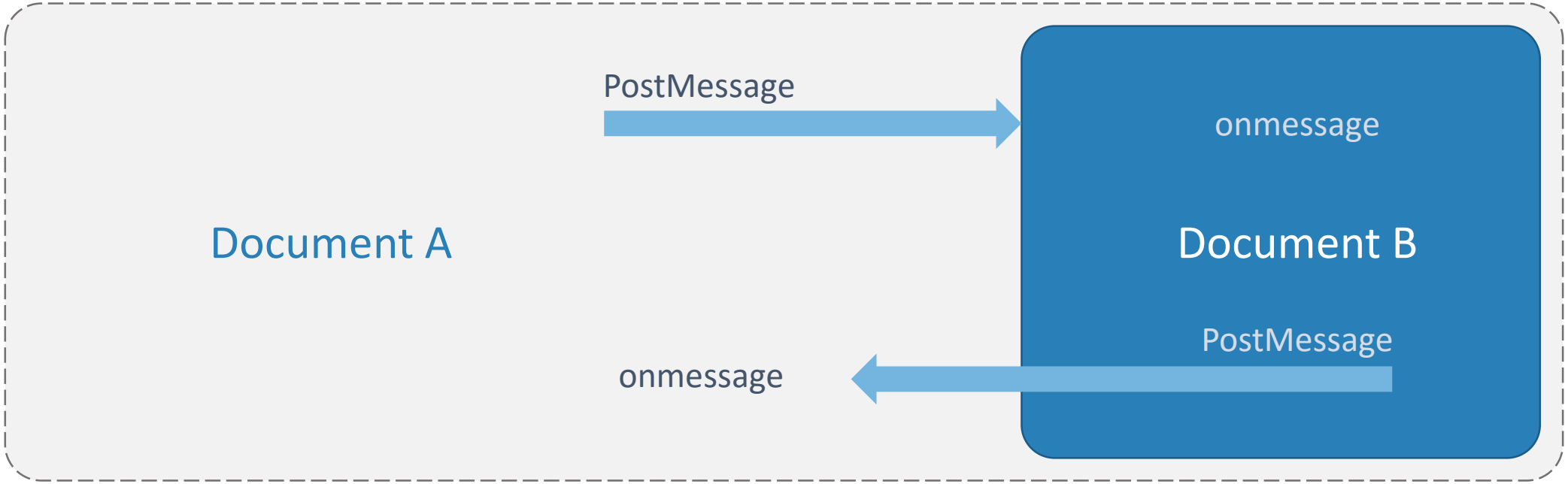
- Same Origin policy prevents communications between different domains:
  - Ajax requests
  - Access to content of iframes etc.

- HTML5 Web Messaging provides the ability to communicate with another origins in 3 simple steps.

postMessage & cross-origin messaging

Steps

1. Get Window object of another document
2. Send data with postMessage method
3. Receive and process message



postMessage & cross-origin messaging

STEP 1. GET WINDOW OBJECT

JAVASCRIPT

```
//Document A

var docB = document.querySelector('iframe').contentWindow;
// or
//var docB = window.open(...);

docB.postMessage('ping', 'documentB.com');
```

STEP 2. GET WINDOW OBJECT

JAVASCRIPT

```
//Document B

window.addEventListener('message', function(event) {

    if(event.origin !== 'documentA.com') {
        return;
    }

    document.querySelector('body').innerHTML = event.data;
    event.source.postMessage(event.data, event.origin);

}, false);
```

STEP 2. SECURITY

- Do not use '*' as url in postMessage when you deal with private data.
- For security reasons always check event.origin against whitelisted domains to ignore messages from malicious sites.
- Check event.data type and content before processing it.

Workers API

Run scripts in background threads

Worker API

Motivation

- JavaScript has single threated nature.
- All asynchronous callbacks are added to the end of execution queue.
- If some block of code runs too long user will feel that browser hangs out.

Worker API

Problem example

JAVASCRIPT

```
function heavyComputingFuntion(i) {
    //e.g. MC simulations here

}

var m = [];

for (var i = 0; i < 1000000; i++) {

    m[i] = heavyComputingFuntion(i);

}
```

Worker API

Simple solution

JAVASCRIPT

```
function makeSimulations(offset, cb) {
    //sorter loop. e.g. 50-100 iterations

    if(finishCondition) {
        cb(m);
    } else {
        setTimeout(makeSimulations, 1, newOffset, cb);
        // or setImmediate(makeSimulations, newOffset, cb)
        // for new browsers

    }

}
```

Worker API

Worker solution. Initial document

JAVASCRIPT

```
var mcWorker = new Worker('MCWorker.js');

mcWorker.postMessage(data);

mcWorker.onmessage = function(event) {
    cb(event.data);
};
```

Another Worker methods and properties:

`postMessage()` – immediately terminates worker.

`onmessage` – event handle. Called when we have uncaught errors in worker.

Worker API

Worker solution. MCWorker.js

JAVASCRIPT

```
function heavyComputingFuntion(i) {
    //MC simulations here
}

onmessage = function(event) {
    var m = [];
    for (var i = 0; i < event.data; i++) {
        m[i] = heavyComputingFuntion(i);
    }

    postMessage(m);
};
```

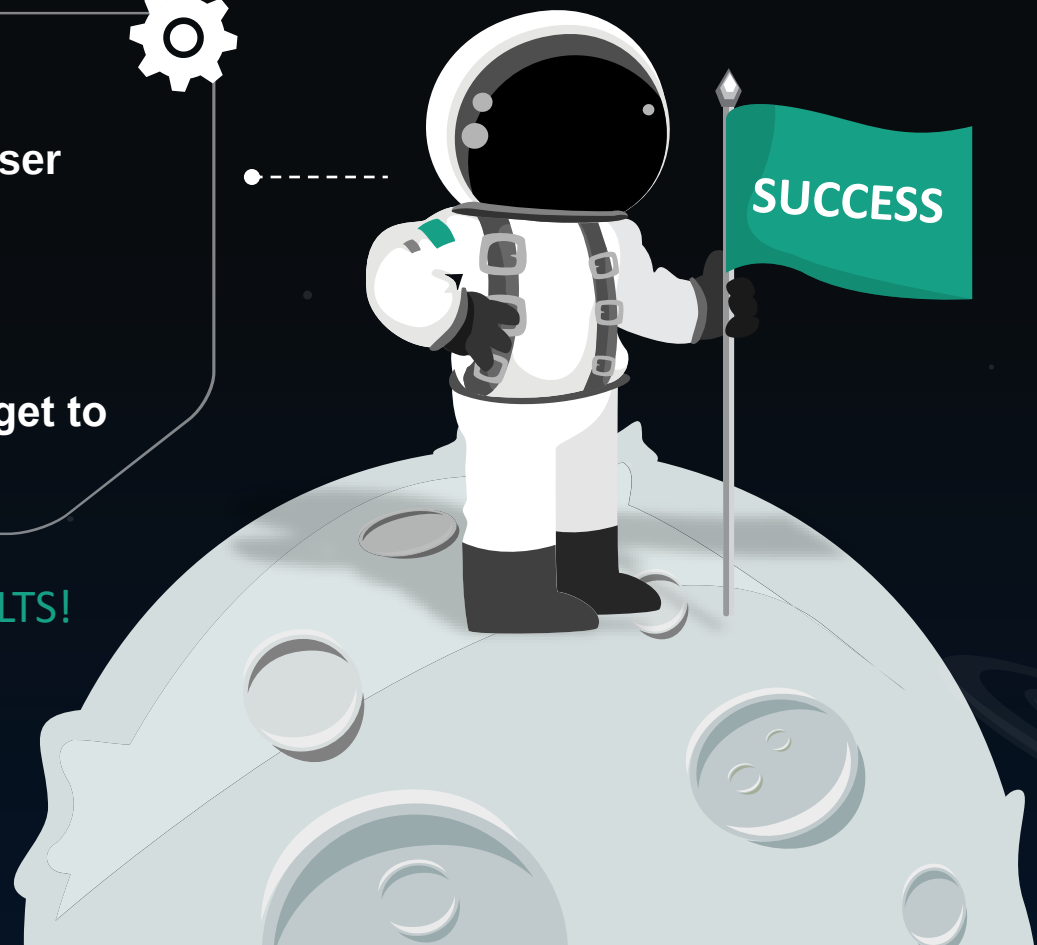
Summary

This is only small part of all browser possibilities

This list is really huge!

It evolves very often, so don’t forget to study new possibilities.

EXPERIMENT AND SHARE YOUR RESULTS!



Any Question ???

THANKS FOR WATCH !!!