Caché User Group (CUG)

Meeting 29/11/2016
Crowne Plaza Antwerp

Agenda

- JavaScript eats the world: introduction and technological overview
  - Intro, overview and current state of the technology
  - Different possible bindings to Caché (CSP, Ajax, REST, Node,js)
- Demo of a Todo app using 4 different approaches/frameworks:
  - « VanillaJS »: plain HTML, JavaScript, REST calls to Caché
  - Sencha ExtJS: monolithic framework, REST calls to Caché using ExtJS data layer
  - AngularJS: monolithic framework, REST calls to Caché using Angular methods
  - React: a view-only framework, Redux/ImmutableJS (data), WebSockets/Ajax/REST mode

Introduction

- What is JavaScript?
- Advantages:
  - Full-Stack: use only one single language at front-end and back-end (and in Caché too?)
  - Is unifying the development world – much easier to hire new people
  - Spans all kinds of development using the same stack: web, (mobile) apps, desktop (e.g. React, React Native, Electron)
  - Everything starts with … Node.js (start using it as your development tool)
- Disadvantages:
  - You'll need to learn it (syntax) BUT you'll find out it has many similarities to COS: flexibility, compactness, performance, …
  - Can be overwhelming at first look, but you'll soon discover that the best choices for building your stack are not so numerous

Did you know ... it covers web, mobile apps, desktop?

- React: web development (Virtual DOM, JSX, components, one-way data flow)
- React Native: write your mobile apps in JavaScript (same JSX syntax, uses native components) - NativeBase (cross-platform) – supports iOS, Android, Universal Windows Platform, Tizen)
- Electron: write desktop apps in JavaScript for Mac, Windows & Linux
- ...

Why do I need a framework?

- You need to write your app, not a framework (don't re-invent the wheel!)
- Work with your team in a standardized way: good frameworks enforce (clear & maintainable) coding patterns
- A framework allows you to use all readily available source code, (debugging) tools, modules, … – provides you with much more options!
- Same goes for « why use Node.js as application server in between? »: use a very large pool of ready-to-use modules for everything you can imagine

JavaScript bindings to Caché

You have different options:

- Using REST calls (only request/response):
  - Caché's built-in REST server running on the CSP gateway
  - Using Node.js (using cache.node e.g. Express module)

  REST requires many server calls, authentication/security is not trivial, work stateless or with sessions?

- Using WebSockets (request/response objects + server push):
  - From CSP (low-level or using socket.io)
  - Using Node.js (using socket.io module, e.g. EWD 3)

  WebSockets allow a direct « open » connection to the server, security is easier, but requires a stable network link (socket.io can degrade gracefully to Ajax calls)

JavaScript bindings to Caché – cont'd

How do I use my SQL, classes, …? What about all my « legacy » code?

No panic! You can re-use your existing code:

- Use CSP pages calling server-side methods
- Use (trivial) wrapper functions in Caché and call them from your Node.js code: you can use everything you like (classes, SQL, …)
- Access your globals directly from JavaScript in Node.js using the ewd-document-store module
- Important for ISC: we need direct JavaScript support in Caché for classes, SQL, … (inside cache.node and as a language inside Caché too). Full-Stack JavaScript development in Caché using one single language!

Node.js binding to Caché – Node.js

- One module (file): cache.node
- Works in-process: architecture of Caché & Node.js (x86/x64) MUST be the same!
- Cache.node version must also match Node.js version ranges (major!)
- Works from version >= 2008.2 onwards (!) - just use latest version
- Can also work in networked mode (Caché & Node.js on different servers)
- What about speed? Excellent results (very fast), however really native Caché performance in JavaScript would even be better: please vote for optimizing Google V8 string handling

Node.js binding to Caché – REST

- On recent Caché versions, use REST Web applications (with CSP gateway)
- But … what about (very!) old Caché versions (pre-2008)? You can still use WebLink as a REST gateway: works perfectly with recent Apache 2.4 builds for Linux & Windows (Windows: Apache Haus & Apache Lounge)

Give your legacy applications a modern facelift!

Asynchronous code & callbacks

What? JavaScript code doesn't execute sequentially?

You'll need to learn to code « event-based »

```
function syncFoo(param) {
  … foo's code
}
syncFoo('foo');
console.log('syncFoo is done!');
```

```
function asyncFoo(param, cb) {
 … foo's code
}
asyncFoo('foo', function () {
  console.log('asyncFoo is done!');
});
console.log('BEFORE asyncFoo is done!');
```

Useful tips

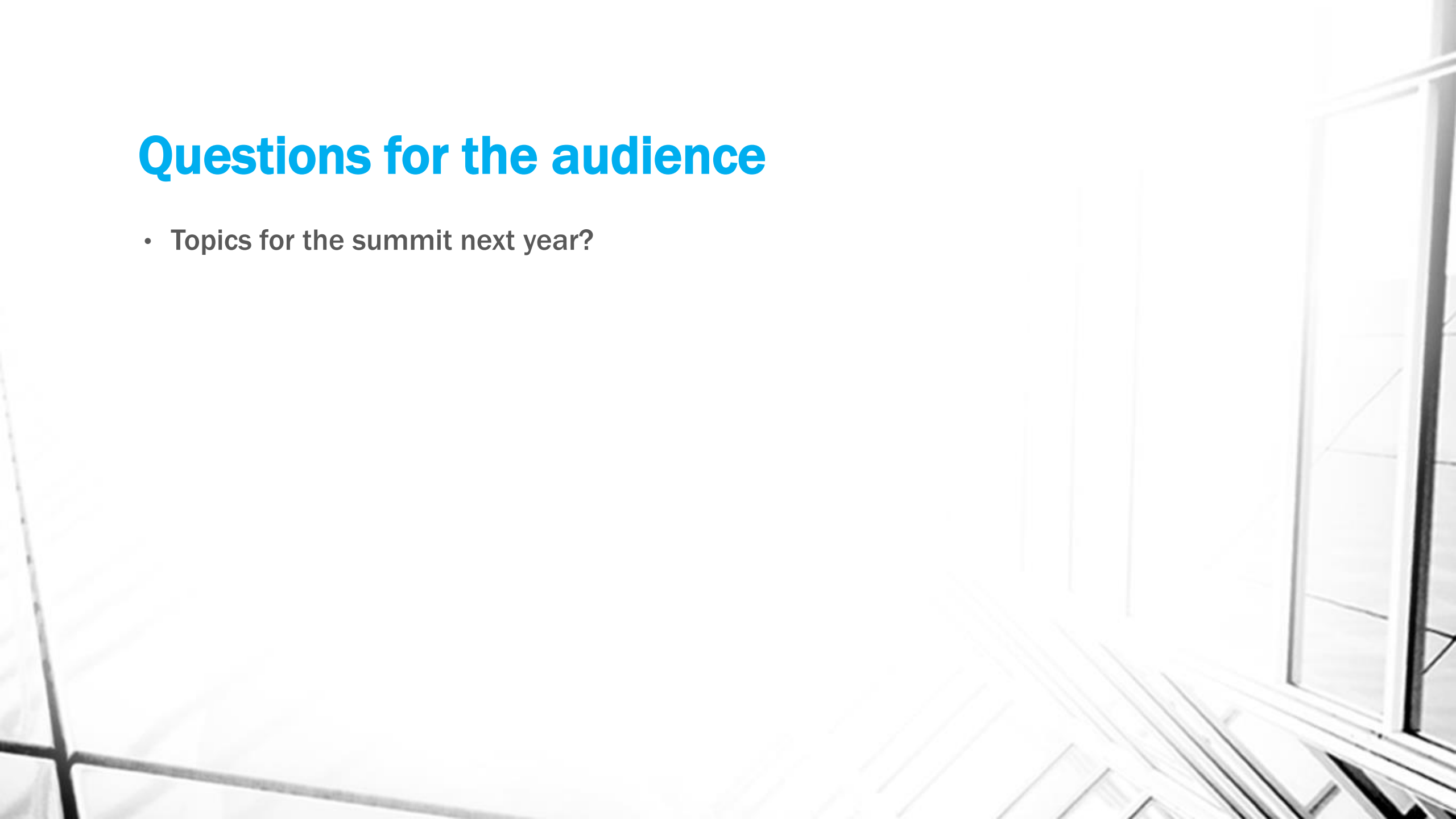
- Use front- and back-end frameworks that take care of most of the plumbing work for you (higher-level abstraction, sessions, security, error handling, ...)
- Consider Node.js (with npm/yarn) for building web applications (provides you with automated tools to create production builds with minification, easily including required modules, development mode with hot reloading)
- For Ajax/REST API's: use fetch (isomorphic) where you can!
- For WebSockets: use socket.io
- The EWD 3 framework (set of Node.js modules) provides Caché binding using:
  - WebSockets (can degrade to Ajax calls) (using EWD.send() method)
  - using only Ajax calls (using the same EWD.send() method)
  - or you can use REST API calls (using browser fetch() method)

Useful links

- The state of JavaScript in 2016
- How it actually feels to write JavaScript in 2016
- How it feels to learn JavaScript in 2016 (but don't take it too seriously!)
- Front-end: React and React Native, AngularJS, Sencha ExtJS, Ember
- Back-end: Node.js, Express (+ EWD 3 to interface to Caché)
- Node.js modules by the numbers

Questions for the audience

- Topics for the summit next year?



CUG Benelux

Blog: http://cug-benelux.be

E-mail: info@cug-benelux.be

Twitter: @cugbenelux

LinkedIn group (discussions): CUG Benelux

Announcements, presentation slides will be posted on the CUG blog. Please also follow us on Twitter, join the LinkedIn group (easily become a member by joining the group) and feel free to discuss topics online! If you have questions, suggestions for the CUG core members, you can also contact us by e-mail.