APIs at scale

Matthew Clark
@matthew1000

Paul Caporn
@drpacman72
Designing for frontends
Designing for change
Designing for scale
Designing for validation
BBC News, the #1 most popular news site in the world

Global visits to News sites, January 2022

Source: Press Gazette / Similarweb
https://tinyurl.com/most-popular-news

BBC iPlayer, the #2 most popular online video service in the UK

% of UK homes consuming TV/film video service

Source: Ofcom
https://tinyurl.com/ofcom-report-2021
### Over 100 APIs power BBC Online

<table>
<thead>
<tr>
<th>API name</th>
<th>primary response type</th>
<th>Compute type</th>
<th>directly powers audience experience?</th>
<th>Publicly advertised?</th>
<th>Open to world?</th>
<th>Access control</th>
<th>Request interface type</th>
<th>Has event-driven?</th>
</tr>
</thead>
<tbody>
<tr>
<td>App Business Layer - News</td>
<td>JSON</td>
<td>FABL</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>None</td>
<td>HTTP</td>
<td>No</td>
</tr>
<tr>
<td>App Business Layer - Sport</td>
<td>JSON</td>
<td>FABL</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>None</td>
<td>HTTP</td>
<td>No</td>
</tr>
<tr>
<td>Account Identity Command Service</td>
<td>JSON</td>
<td>EC2</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Certificate</td>
<td>HTTP</td>
<td>No</td>
</tr>
<tr>
<td>Account token</td>
<td>JSON</td>
<td>EC2</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Cookie</td>
<td>HTTP</td>
<td>No</td>
</tr>
<tr>
<td>APPW</td>
<td>JSON</td>
<td>S3</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>IAM</td>
<td>S3</td>
<td>Yes</td>
</tr>
<tr>
<td>Archive Services API</td>
<td>JSON</td>
<td>Serverless</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Key</td>
<td>HTTP</td>
<td>No</td>
</tr>
<tr>
<td>Ares</td>
<td>JSON</td>
<td>EC2</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Certificate</td>
<td>HTTP</td>
<td>No</td>
</tr>
<tr>
<td>Article domain</td>
<td>JSON</td>
<td>FABL</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Certificate</td>
<td>HTTP</td>
<td>No</td>
</tr>
<tr>
<td>Audiences Portal API</td>
<td>JSON</td>
<td>EC2</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Certificate</td>
<td>HTTP</td>
<td>No</td>
</tr>
<tr>
<td>BBC Ideas</td>
<td>JSON</td>
<td>Morph</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Certificate</td>
<td>HTTP</td>
<td>Yes</td>
</tr>
<tr>
<td>BBC Live</td>
<td>JSON</td>
<td>Morph</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>None</td>
<td>HTTP</td>
<td>Yes</td>
</tr>
<tr>
<td>Belfrage</td>
<td>Multiple</td>
<td>EC2</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>None</td>
<td>HTTP</td>
<td>Yes</td>
</tr>
<tr>
<td>Binary Store</td>
<td>Multiple</td>
<td>EC2</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>None</td>
<td>HTTP</td>
<td>No</td>
</tr>
<tr>
<td>Blocks (like Arts)</td>
<td>JSON</td>
<td>Morph</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Certificate</td>
<td>HTTP</td>
<td>Yes</td>
</tr>
<tr>
<td>Broadcast interactive data</td>
<td>JSON</td>
<td>EC2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Certificate</td>
<td>HTTP</td>
<td>Yes</td>
</tr>
<tr>
<td>Camino (News recs)</td>
<td>JSON</td>
<td>EC2</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Certificate</td>
<td>HTTP</td>
<td>Yes</td>
</tr>
<tr>
<td>Comments</td>
<td>JSON</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Token</td>
<td>HTTP</td>
<td>No</td>
</tr>
<tr>
<td>Content Summaries</td>
<td>JSON</td>
<td>FABL</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Certificate</td>
<td>HTTP</td>
<td>No</td>
</tr>
<tr>
<td>Cosmos API</td>
<td>JSON</td>
<td>EC2</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Certificate</td>
<td>HTTP</td>
<td>No</td>
</tr>
<tr>
<td>Counting Service</td>
<td>JSON</td>
<td>Serverless</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>None</td>
<td>HTTP</td>
<td>Yes</td>
</tr>
</tbody>
</table>
How many are serverless?

- Server: 67%
- Serverless: 33%

Event or request-driven?

- Event-driven: 33%
- Request-driven: 67%

What means of access control?

- Mutual TLS: 52%
- Access key: 14%
- Public: 30%
- Other: 4%

What response format?

- JSON: 62%
- XML: 7%
- GraphQL: 4%
- Other: 27%
Designing for frontends
Micro-frontend API for TV’s

TV eco-system is fragmented (long tail)
Code using the API had sprawled...

Monolithic, spaghetti code
Micro-frontend API for TV’s

• Launch a kids app!
• Renew and iterate on the UI!
• But unclear boundaries / lack of a clear API
Micro-frontend API for TV’s

“One Love” project – enabling decoupled development
Micro-frontend API for TV’s

Component API – A small surface, clear separation of concerns

```javascript
const apiFactory = () => {

    // Add (initialisation) logic and internal state here

    const componentRuntimeAPI = {
        blur: () => { /* called when component loses focus */ },
        destroy: () => { /* called when component is discarded */ },
        focus: () => { /* called when component becomes focused */ },
        handleKeyDown: (keyCode) => { /* called with keyCode for keyDown event */ },
    }

    return componentRuntimeAPI
}
```
Micro-frontend API for TV’s

Enabler for development of rich, data driven UI components in isolation

const MyComponent = {
  id: 'myComponent',
  render: () => makeRequestTo('my/new/component')
    .then((body) => {
      return {
        html: JSON.parse(body).html,
        apiFactory: () => {
          return window.components[id](makeRequestTo, eventHandler)
        }
      }
    })
}
Micro-frontend API for TV’s

Enabler for development of rich, data driven UI components in isolation

```javascript
const MyComponent = {
  id: 'myComponent',
  render: () => makeRequestTo('my/new/component')
    .then((body) => {
      return {
        html: JSON.parse(body).html,
        apiFactory: () => {
          return window.components[id](makeRequestTo, eventHandler)
        }
      }
    })
}
```
Micro-frontend API for TV’s

Enabler for development of rich, data driven UI components in isolation

```javascript
const MyComponent = {
  id: 'myComponent',
  render: () => makeRequestTo('my/new/component')
    .then((body) => {
      return {
        html: JSON.parse(body).html,
        apiFactory: () => {
          return window.components[id](makeRequestTo, eventHandler)
        }
      }
    })
}
```
const MyComponent = {
  id: 'myComponent',
  render: () => makeRequestTo('my/new/component')
    .then((body) => {
      return {
        html: JSON.parse(body).html,
        apiFactory: () => {
          return window.components[id](makeRequestTo, eventHandler)
        }
      }
    })
}

- There is an integration per component in the main TV client application.
- By convention, the HTML for components delivered from the backend includes a <script> tag that executes an IIFE exposing a constructor for the runtime API.
- Components are passed local client capabilities via the constructor.
- Components issue commands to the host by emitting (uni-directional) events via a passed in `eventHandler`.
- The BFF services have versioned endpoints (HTTP APIs) to facilitate making breaking changes to the integration/runtime API.
**Live Reporting**

Rex Bryson

**13:35**

**Quarantined balls, please**

Sonia Ousey
BBC Sport at Wimbledon

Even the used tennis balls are subject to Covid restrictions this year.

Normally they come straight from the court before being sold to the public for fans' chance to get their hands on a bit of yellow fluff hit by Roger Federer et al.

But this time they have to be put in quarantine for "a few days".

So by my reckoning, the ones on sale today could be ones hit by Andy Murray last week (maybe).
APIs doing one thing well

Considering the...
- Data
- Scaling
- Restrictions
- Security
- Clients

Website
- Video page
- Home page
- Article page

App
- Curation
- Article
- Video

Backend for frontend
Thanks, X-ray
Multiple teams ➔ inconsistency

Have conventions
Designing APIs for change
IBL – Product as an API

The iPlayer Business Layer (IBL) API is...

• The source of truth for all Product concepts
• The source of truth for content object presence and order
• Not opinionated about the UI

The personalised version of the API is exposed over GraphQL
IBL – Product as an API

Every property exposed has a cost once published

IBL – Product as an API

• Know the purpose of your API
• Design explicitly for your clients use cases
• Limit your supported consumers to those use cases
• Persisted GraphQL queries can help manage evolution!
Handling API change

```
{
    "id": "urn:bbc:article:123",
    "firstPublished": "2022-03-27T11:49:00.000Z",
    "title": "Duck swims around the world",
    "category": "nature",
    "categories": ["nature", "travel"],
}
```

```
<image deprecated="true" deprecated_since="2014-07-09" replaced_by="mixin=images" url="http://..."/>
```

image: String @deprecated(reason: "image is deprecated. Use imageUrl instead.")

Actual BBC API example

GraphQL schema
API versioning

Version the whole API...

Or just one end-point...

Example from AWS S3 API

- ListBuckets
- ListMultipartUploads
- ListObjects
- ListObjectsV2
- ListObjectVersions
- ListParts

https://api.twitter.com/1.1/statuses/lookup.json
https://api.twitter.com/2/tweets
Many clients

From: Joe Developer
To: User d-list
Subject: New API Version

You must upgrade!

The original API will be switched working indefinitely.
Creating a facade

Client → API v1 → API v1 facade → API v2 → Client

Client
Client
Client
Client
Client

API v1 facade
Know who’s involved

- Your developers
- Client’s developers
- Your stakeholders
- Client’s stakeholders

Suggested priority
Designing APIs for **scale**
A broadcast Denial of Service attack
Serverless Launch

- Leverage CDN edge functions
- Statically pre-published responses per device to S3
- Device categorization in lambda@edge
- Augment with dynamic values
- But what about the next hop...?
Fallbacks - failure as a first class citizen

• Fallbacks - planning for failure as a first class part of the API design, not an afterthought

• What do fallbacks need to do?
  • Provides schema compliant lighter responses
  • Supports context sensitivity
  • Enable scale by offloading

• Trade offs, trade offs everywhere
  • Availability over degraded features and timeliness of content
  • Polling vs event driven publishing
Serving Fallbacks
Serving Fallbacks
Serving Fallbacks
Serving Fallbacks
Serving Fallbacks
A third of the BBC’s pages are rendered using serverless (including the APIs that power them)
If using AWS, the CDK patterns make serverless API development straightforward.

https://github.com/cdk-patterns/serverless
Low-code / no-code APIs

https://github.com/cdk-patterns/serverless/blob/main/the-dynamo-streamer/README.md
Scaling instantly when you need it

Lambda response times

User → Website → Application API → Domain API → Database → Server

- Serverless Website
- Serverless Application API
- Serverless Domain API
- Server
- Database

Report changed content

- cache
- cache
- cache

Millisecs
- Duration p50
- Duration p75
- Duration p99
Anime: How Japanese animation has taken the West by storm

By Becky Padgham & Ian Youngs
BBC News

2 days ago

Japanese animation Jujutsu Kaisen 0 has become a surprise box office hit, as anime enjoys a new global popularity in cinemas and on streaming services.

Last weekend, an animated Japanese high school student who is haunted by the spirit of his childhood sweetheart was second only to The Batman in the UK box office chart, and beat Catherine Tate and Sir Mark Rylance, who starred in other new releases.

Jujutsu Kaisen 0 - a prequel to a successful streaming TV series, itself based on popular manga comics - has become the latest successful anime export.
Manifest API

```json
{
    "id" : "/news/123456",
    "lastUpdated" : 1648498767000,
    "title" : "Parrot learns seven times table"
},
{
    "id" : "/news/234567",
    "lastUpdated" : 1648496828000,
    "title" : "Studies show fish happier on Fridays"
},
{
    "id" : "/news/345678",
    "lastUpdated" : 1648497939000,
    "title" : "Herd of elephants found in Norway"
}

One fast-updating, minimally cached API can provide references to another API with more immutable data

```json
{
    "id" : "/news/123456",
    "lastUpdated" : 1648498767000,
    "body" : "A parrot in Italy has learned to count in sevens. Billy, a female macaw, regularly recites the seven times table up to 105. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad"
}
```
Designing APIs for validation
Schemas for the win!

• Provide a contract
• Enable stubbing by client teams
• Enable value adds to be built...
Unreliable Upstream

Like your upstream services but worse...

• Data that meets their API
• Random response times and occasional errors
• Response profiles with configured delays and error rates.
Designing for **frontends**

Designing for **change**

Designing for **scale**

Designing for **validation**
Summary

Focus on doing one thing well

Serverless helps APIs with speed and scale

Know your clients

Treat failure as a first-class citizen

Matthew Clark
@matthew1000

Paul Caporn
@drpacman72