

W E L C O M E
TO *Fabulous*
TECHORAMA
ANTWERP

Building Offline Capable PWA's

Yves Goeleven

Room 1

11:30



Yves Goeleven



Solution Architect

- Goeleven BV, www.goeleven.com
- Advisory consulting & training
 - Currently @ Q-Park via Cegeka
- Products
 - www.clubmanagement.io
 - www.messagehandler.net
 - Dish
- Board member @ Basket Lummen
- Co-founder & board member @ AZUG.be, 9 x former Azure MVP



DEMO

Introduction

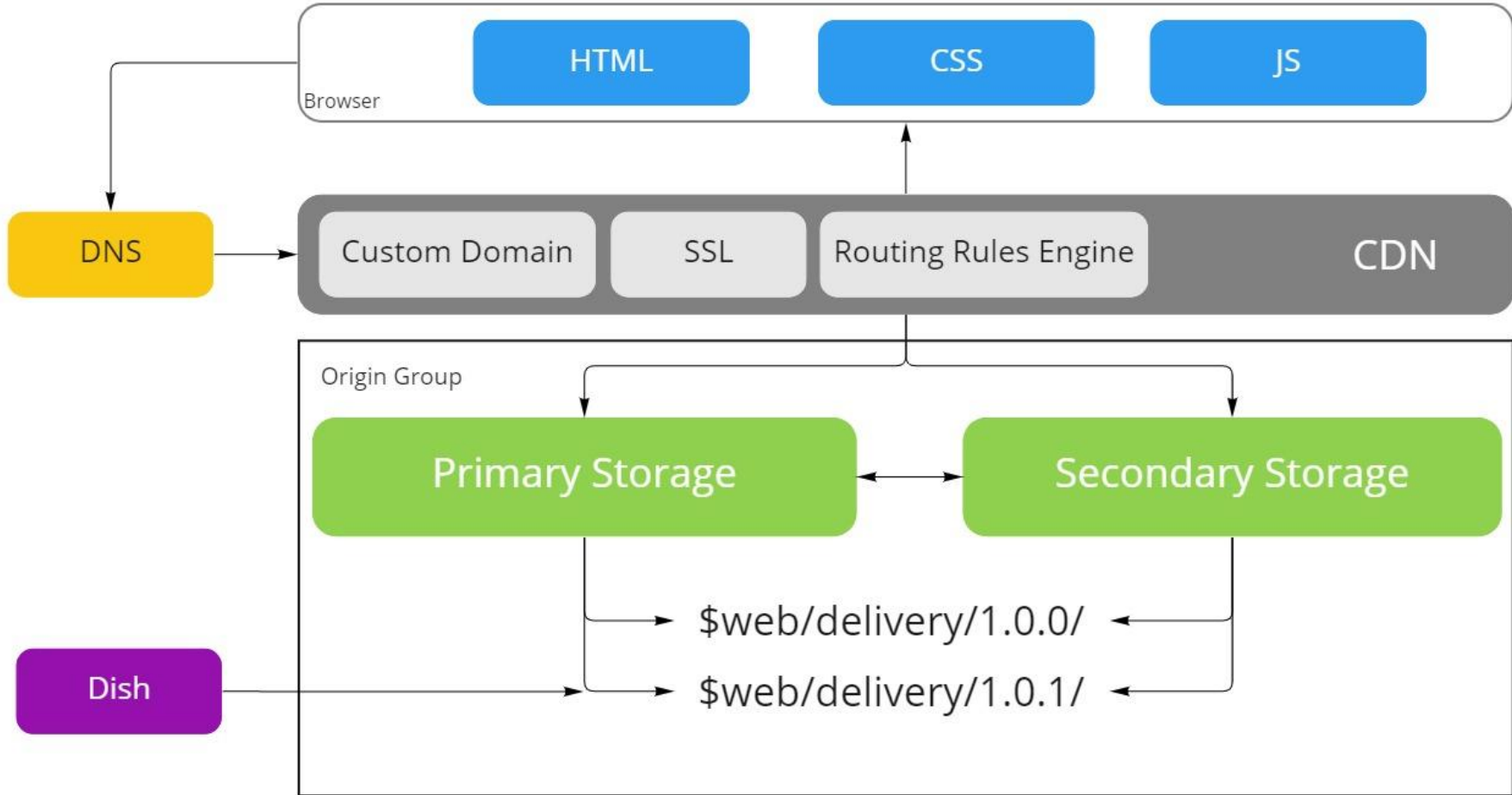
Architecture

Online or offline

Not all data is equal

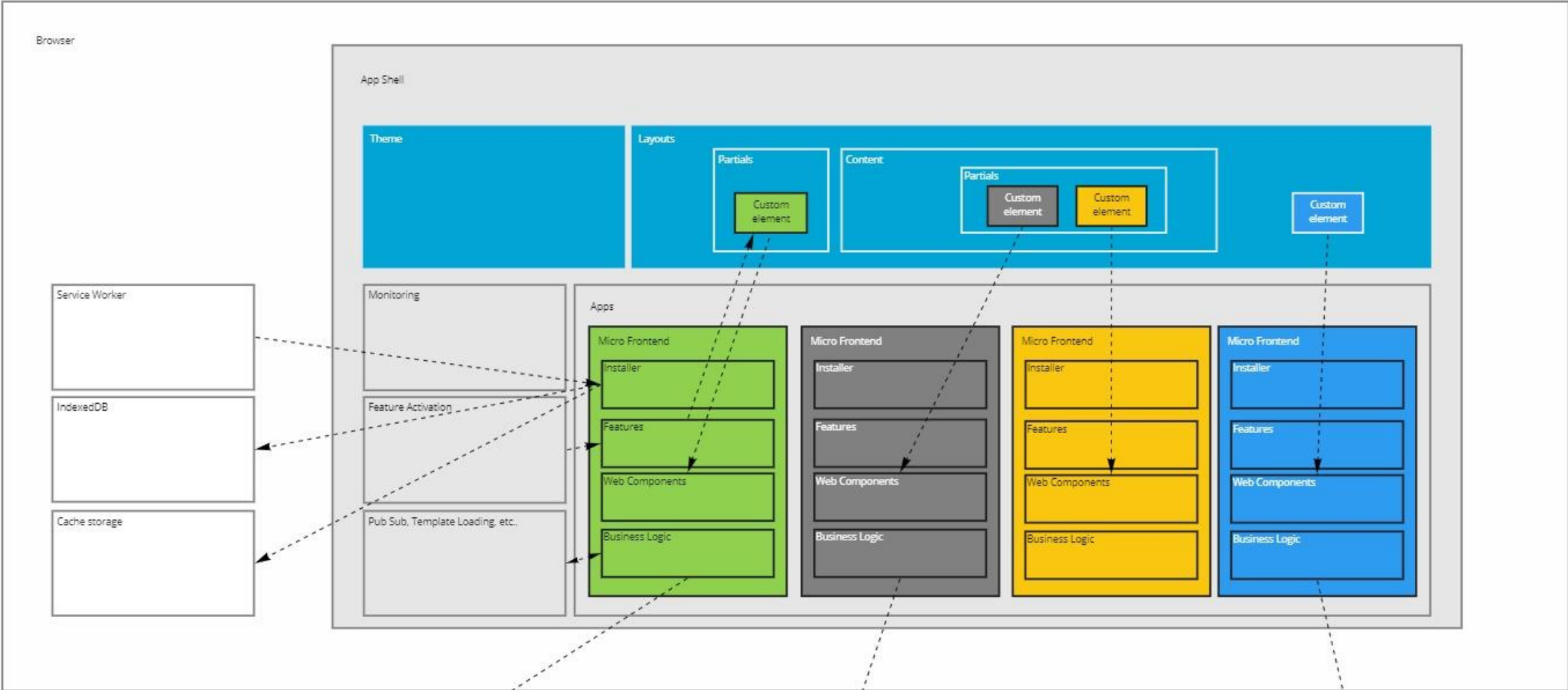
Hosted on CDN

Backed by geo redundant Azure Storage



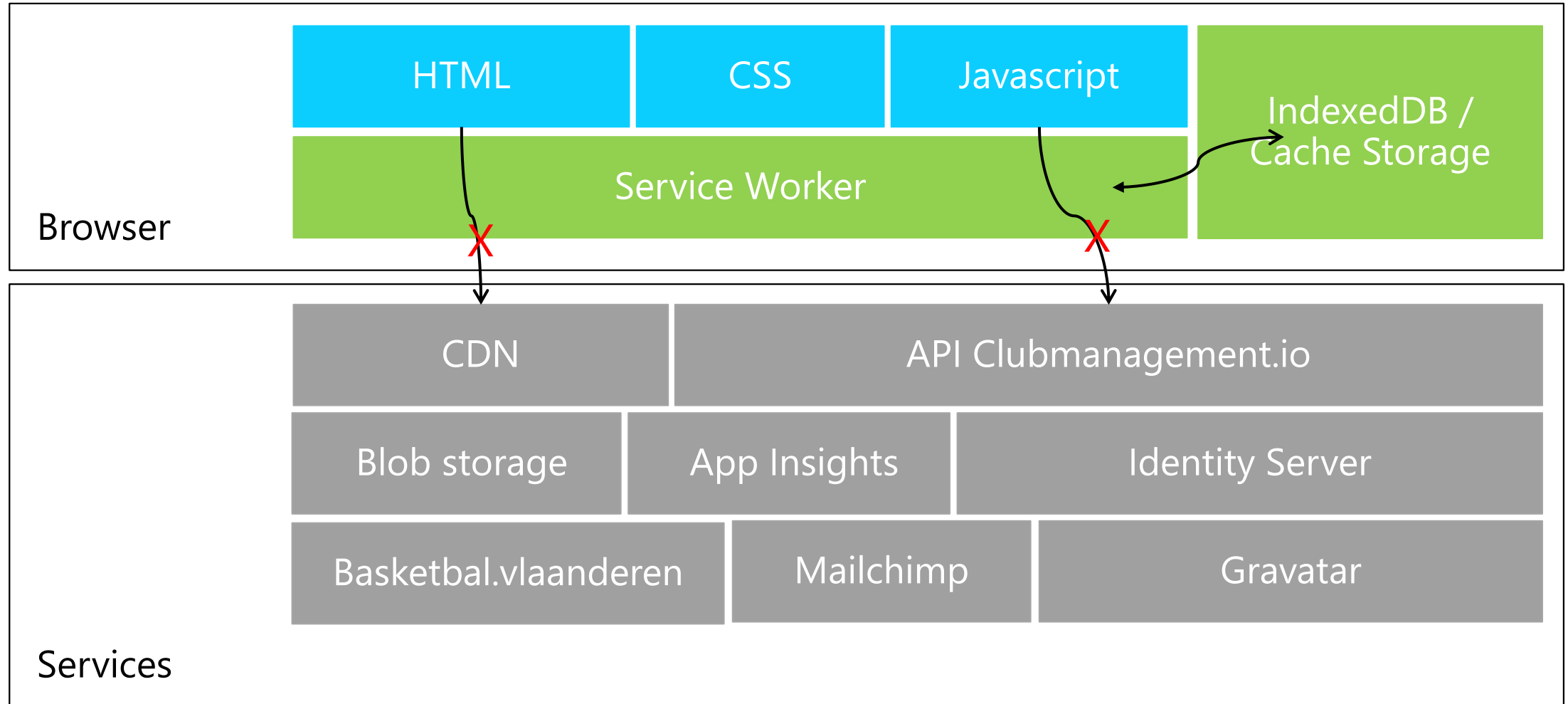
Front end architecture

App Shell & Micro Frontends (HTML, CSS & JS)



How it works, even when offline

Service Worker intercepts and serves from IndexedDB or Cache storage



Introduction

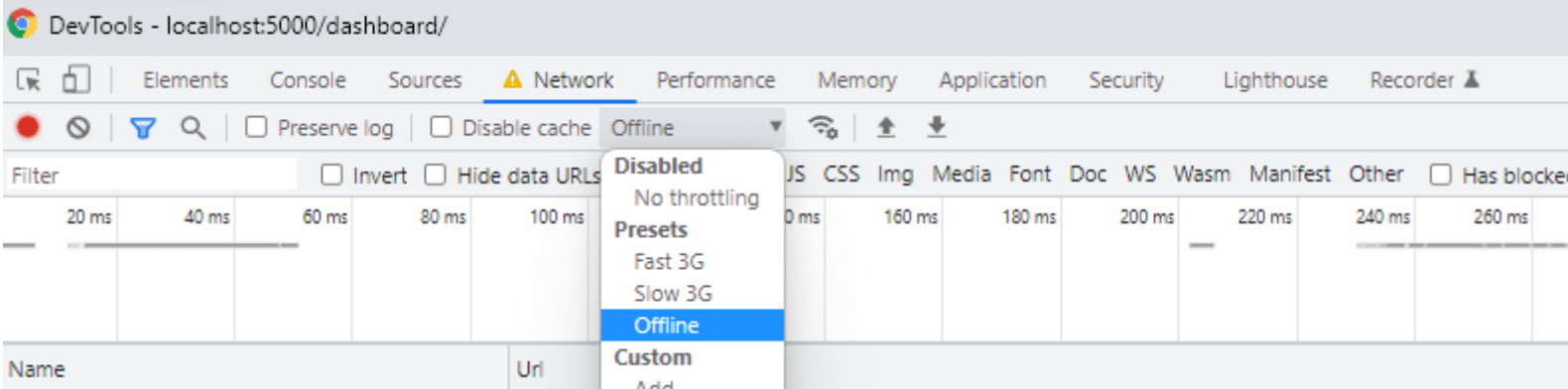
Architecture

Online or offline

Not all data is equal

Going offline

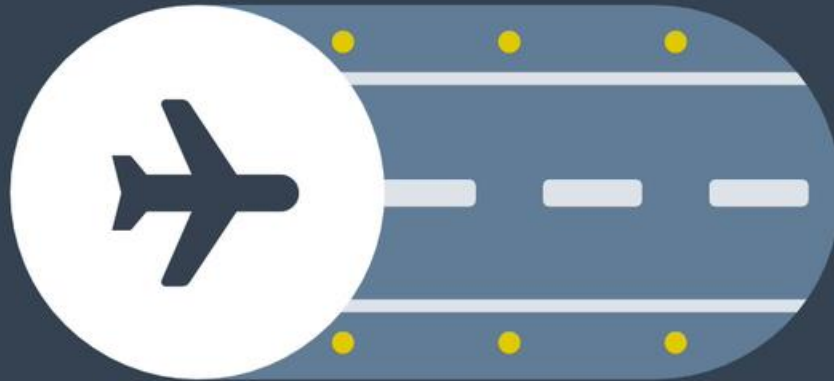
Dev tools, network inspector & console output



The screenshot shows the Chrome DevTools Network tab. The 'Offline' mode is selected in the throttling dropdown menu. The console output shows the following messages:

```
workbox Router is responding to: https://login-test.clubmanagement.io/.well-known/openid-configuration
workbox Using StalewhileRevalidate to respond to 'https://login-test.clubmanagement.io/.well-known/openid-configuration'
workbox Network request for 'https://login-test.clubmanagement.io/.well-known/openid-configuration' threw an error. TypeError: Failed to fetch
    at StrategyHandler.js:159:39
    at StrategyHandler.fetch (StrategyHandler.js:122:31)
    at StrategyHandler.fetchAndCachePut (StrategyHandler.js:205:37)
    at StalewhileRevalidate._handle (StalewhileRevalidate.js:75:14)
    at StalewhileRevalidate._getResponse (Strategy.js:143:35)
Uncaught (in promise) TypeError: Failed to fetch
    at StrategyHandler.js:159:39
    at StrategyHandler.fetch (StrategyHandler.js:122:31)
    at StrategyHandler.fetchAndCachePut (StrategyHandler.js:205:37)
    at StalewhileRevalidate._handle (StalewhileRevalidate.js:75:14)
    at StalewhileRevalidate._getResponse (Strategy.js:143:35)
```

`window.navigator.onLine === ?`



Window.navigator.onLine

!Online == not on the internet, but online != on the internet

- E.g Virtual network adapter may result in network connection, but no internet

```
class Shell{

  constructor(){
    this.onLine = false;
    window.addEventListener('online', (e)=> { this.checkIsOnline(); });
    window.addEventListener('offline', (e)=> { this.checkIsOnline(); });
  }

  async checkIsOnline() {
    if(!window.navigator.onLine){ this.onLine = false; }
    else
    {
      const url = new URL(window.location.origin);
      url.searchParams.set('isonline', Math.random().toString(36).substring(2, 15));
      try{
        const response = await fetch(url.toString(), { method: 'HEAD' })
        this.onLine = response.ok;
      }
      catch{
        this.onLine = false;
      }
    }
    this.topics.publish(this.onLine ? "online" : "offline");
  }
}
```


Introduction

Architecture

Online or offline

Not all data is equal

Not all requests & responses are equal

Unstructured



↑
Needs versioning

Structured

Command	Event	State
<ul style="list-style-type: none">- Intent- Future- Uncertain outcome- Least stable	<ul style="list-style-type: none">- Fact- Past- Certain outcome- Most stable	<ul style="list-style-type: none">- Properties- Certain point in time- Evolving- Cacheable for a while



↑
Hard to cache



↑
Cacheable forever



↑
Cacheable for a while

Cache storage

Cache for request/response pairs

- Low level API
- Caching (unstructured) responses
- Directory structure
- Key / Response pairs
 - Where key is the request

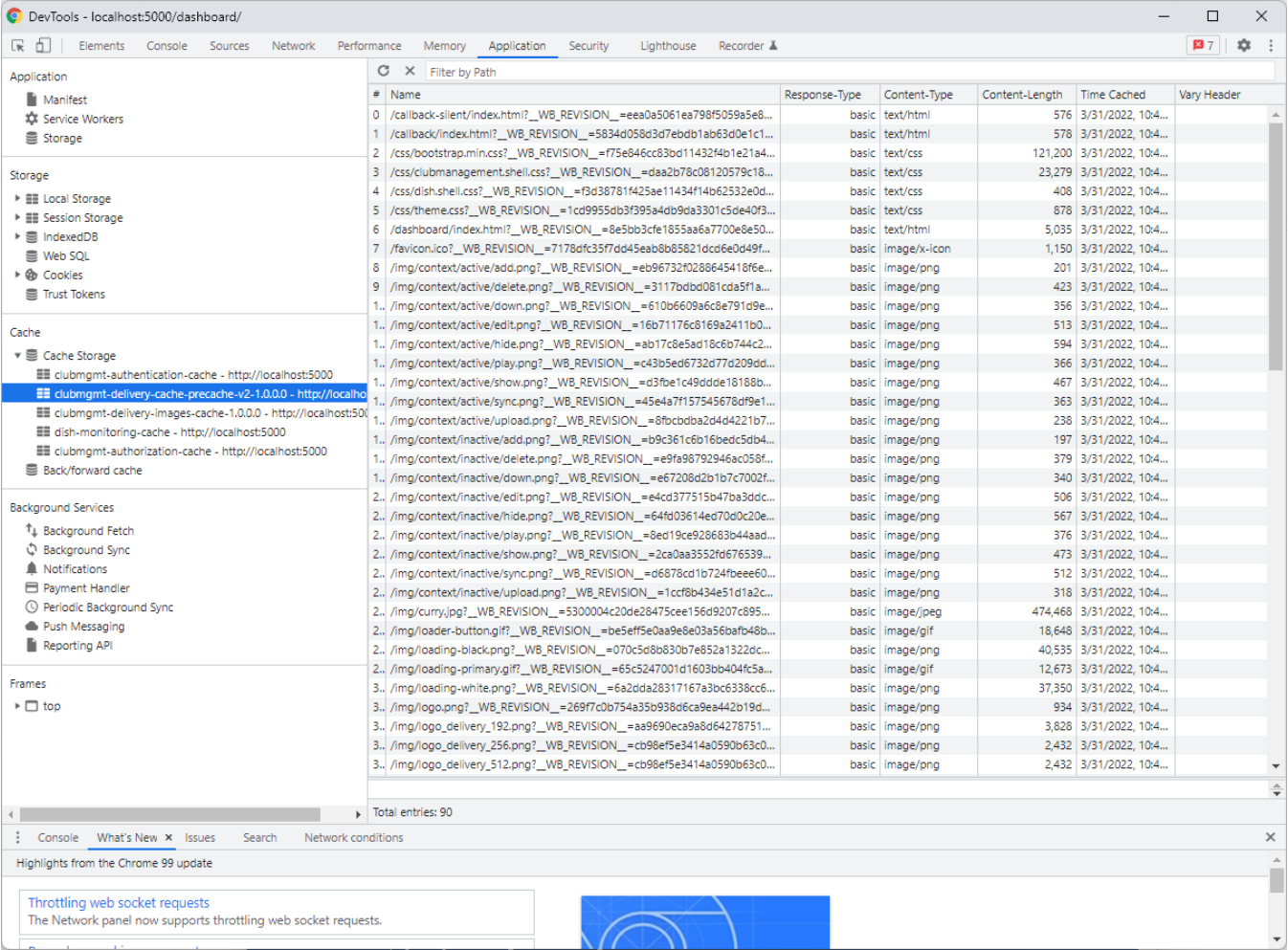
IndexedDB

A database built into every browser

- Low level API
- Caching structured data
- Object database
- Key / Object pairs
 - Key is a property
 - Objects must support `structuredClone()`
- Transactional
- Supports indexes
 - Cursor based

Inspecting cache storage & indexedDB

Dev tools, application, storage / cache



Working with requests & responses

Workbox

Files

CORS

OAuth

Queueing
Requests

Service Worker

Background worker for web pages

- Script runs on a background thread (even when browser is closed)
- Exposes a few events
 - **Install**: Fired whenever the worker gets installed (**first request**)
 - **Activate**: Fired whenever a loaded page connects to the worker
 - **Fetch**: **Fired every time a connected client performs a request**
 - **Message**: Communication between frontend thread and background worker
 - **Sync**: Fired after device comes online
 - **Push**: Fired when device receives a push notification
- You need to write code to handle all those events

```
self.addEventListener('fetch', function(event){  
  // your code here  
});
```

Google workbox

A set of common patterns for Service Worker

- A set of javascript libraries that help you implement service workers

```
importScripts('https://storage.googleapis.com/workbox-cdn/releases/6.0.2/workbox-sw.js');

workbox.setConfig({ debug: true });

workbox.core.setCacheNameDetails({
  prefix: 'clubmgmt-delivery-cache',
  suffix: '1.0.0.0'
});
```

- Available modules
 - **Precaching**: preload files on install
 - **Request routing**: configure how to handle specific web requests
 - **Runtime caching strategies**: caching strategies
 - **Expiration**: remove cached entries
 - **BackgroundSync**: resubmit requests after coming online

Offline caching

Available strategies

- Pre-caching: download and cache responses up front (**first request**)
- Runtime caching: download and cache response per child request
 - **Stale-While-Revalidate**: Cache, network update after response
 - **Cache First**: Cache falling back to network
 - **Network First**: Network falling back to cache
 - **Network only**: Never from cache, always from network
 - **Cache only**: Always from cache, never from network
 - **Custom**: Build your own
- Plugins:
 - **Background-sync**: queues up failed requests, to be synced later
 - **Broadcast-update**: inform open windows when cached response gets updated
 - **Expiration**: Limit how long and how many items get cached
 - **Range-request**: Supports partial responses

Working with requests & responses

Workbox

Files

CORS

OAuth

Queueing
Requests

'Installing' files

Precaching: Download & cache all files of the web app

- Service worker 'Install' event
 - All essential files for the app will be downloaded and cached on first request
 - Route with a 'Cache-first' policy: Cache Falling Back to Network (**No autoupdate!**)

```
workbox.precaching.precacheAndRoute(self.__precacheManifest || []);
```

- Files must be versioned
 - Revision hashes (generated by Dish postprocessing step)
 - Service worker file must change for changes to take effect: build revision

```
self.__precacheManifest = [  
  { url: '/index.html', revision: '16a3cdb338289d....74564ccd3db2430bac' },  
  { url: '/css/bootstrap.min.css', revision: '5a3d8c05785485d3....8b5bd7b0f3168fff1bd9a' },  
  { url: '/css/console.css', revision: '900797671c753ea9b421....f1db2874f32d6264996801' },  
  ....  
]
```

- Your app shell can run offline now, future pages served from cache

Inspecting installed files

Dev tools, application storage

The screenshot shows the Chrome DevTools Application panel for a local development environment. The left sidebar displays a tree view of the application's storage, including Manifest, Service Workers, Storage, Cache, Background Services, and Frames. The main panel shows a list of installed files with columns for Name, Response-Type, Content-Type, Content-Length, Time Cached, and Vary Header. The file `clubmgmt-delivery-cache-precache-v2-1.0.0.0` is highlighted in blue.

#	Name	Response-Type	Content-Type	Content-Length	Time Cached	Vary Header
0	/callback-silent/index.html?__WB_REVISION__=eea0a5061ea798f5059a5e8...	basic	text/html	576	3/31/2022, 10:4...	
1	/callback/index.html?__WB_REVISION__=5834d058d3d7ebdb1ab63d0e1c1...	basic	text/html	578	3/31/2022, 10:4...	
2	/css/bootstrap.min.css?__WB_REVISION__=f75e846cc83bd114324b1e21a4...	basic	text/css	121,200	3/31/2022, 10:4...	
3	/css/clubmanagement.shell.css?__WB_REVISION__=daa2b78c08120579c18...	basic	text/css	23,279	3/31/2022, 10:4...	
4	/css/dish.shell.css?__WB_REVISION__=f9d38781425ae11434f14b62532e0d...	basic	text/css	408	3/31/2022, 10:4...	
5	/css/theme.css?__WB_REVISION__=1cd9955db3f395a4db9da3301c5de40f3...	basic	text/css	878	3/31/2022, 10:4...	
6	/dashboard/index.html?__WB_REVISION__=8e5bb3cf1855aa6a7700e8e50...	basic	text/html	5,035	3/31/2022, 10:4...	
7	/favicon.ico?__WB_REVISION__=7178dfc35f7dd45eab8b85821dcd6e0d49f...	basic	image/x-icon	1,150	3/31/2022, 10:4...	
8	/img/context/active/add.png?__WB_REVISION__=eb96732f0286645418f6e...	basic	image/png	201	3/31/2022, 10:4...	
9	/img/context/active/delete.png?__WB_REVISION__=3117b0bd081cda5f1a...	basic	image/png	423	3/31/2022, 10:4...	
1..	/img/context/active/down.png?__WB_REVISION__=610b6609a6c8e791d9e...	basic	image/png	356	3/31/2022, 10:4...	
1..	/img/context/active/edit.png?__WB_REVISION__=16b71176c8169a2411bd...	basic	image/png	513	3/31/2022, 10:4...	
1..	/img/context/active/hide.png?__WB_REVISION__=ab17c8e5ad18c6b744c2...	basic	image/png	594	3/31/2022, 10:4...	
1..	/img/context/active/play.png?__WB_REVISION__=c43b5ed6732d77d209dd...	basic	image/png	366	3/31/2022, 10:4...	
1..	/img/context/active/show.png?__WB_REVISION__=d3fbc1c49d0de18188b...	basic	image/png	467	3/31/2022, 10:4...	
1..	/img/context/active/sync.png?__WB_REVISION__=45e4a7f1f57545678df9e1...	basic	image/png	363	3/31/2022, 10:4...	
1..	/img/context/active/upload.png?__WB_REVISION__=8f0c0ba2d44221b7...	basic	image/png	238	3/31/2022, 10:4...	
1..	/img/context/inactive/add.png?__WB_REVISION__=b9c361c6b16bedc5db4...	basic	image/png	197	3/31/2022, 10:4...	
1..	/img/context/inactive/delete.png?__WB_REVISION__=e9fa98792946ac058f...	basic	image/png	379	3/31/2022, 10:4...	
1..	/img/context/inactive/down.png?__WB_REVISION__=e67208d2b1b7c7002f...	basic	image/png	340	3/31/2022, 10:4...	
2..	/img/context/inactive/edit.png?__WB_REVISION__=e4cd377515b47ba3ddc...	basic	image/png	506	3/31/2022, 10:4...	
2..	/img/context/inactive/hide.png?__WB_REVISION__=64fd03614ed70d0c20e...	basic	image/png	567	3/31/2022, 10:4...	
2..	/img/context/inactive/play.png?__WB_REVISION__=8ed19ce928683b44aad...	basic	image/png	376	3/31/2022, 10:4...	
2..	/img/context/inactive/show.png?__WB_REVISION__=2ca0aa3552fd676539...	basic	image/png	473	3/31/2022, 10:4...	
2..	/img/context/inactive/sync.png?__WB_REVISION__=d6878cd1b724fbee60...	basic	image/png	512	3/31/2022, 10:4...	
2..	/img/context/inactive/upload.png?__WB_REVISION__=1cctfb43ae51d1a2c...	basic	image/png	318	3/31/2022, 10:4...	
2..	/img/curry.jpg?__WB_REVISION__=5300004c20de28475cee156d9207c695...	basic	image/jpeg	474,468	3/31/2022, 10:4...	
2..	/img/loader-button.gif?__WB_REVISION__=be5eff5e0aa9e8e03a56baf048b...	basic	image/gif	18,648	3/31/2022, 10:4...	
2..	/img/loading-black.png?__WB_REVISION__=070c5d8b830b7e852a1322dc...	basic	image/png	40,535	3/31/2022, 10:4...	
2..	/img/loading-primary.gif?__WB_REVISION__=65c5247001d1603bb0404f5a...	basic	image/gif	12,673	3/31/2022, 10:4...	
3..	/img/loading-white.png?__WB_REVISION__=6a2dda28317167a3bc6338cc6...	basic	image/png	37,350	3/31/2022, 10:4...	
3..	/img/logo.png?__WB_REVISION__=269f7c0b754a35b938d6ca9ea442b19d...	basic	image/png	934	3/31/2022, 10:4...	
3..	/img/logo_delivery_192.png?__WB_REVISION__=aa9690eca9a8d64278751...	basic	image/png	3,828	3/31/2022, 10:4...	
3..	/img/logo_delivery_256.png?__WB_REVISION__=cb98ef5e3414a0590b63c0...	basic	image/png	2,432	3/31/2022, 10:4...	
3..	/img/logo_delivery_512.png?__WB_REVISION__=cb98ef5e3414a0590b63c0...	basic	image/png	2,432	3/31/2022, 10:4...	

Total entries: 90

Console: What's New | Issues | Search | Network conditions

Highlights from the Chrome 99 update

Throttling web socket requests
The Network panel now supports throttling web socket requests.

Workbox will serve files 'Cache first'

Request routing has its differences though...

- File paths behave differently, `'/dashboard' != '/dashboard/'`
- Capitalization matters (case sensitive like linux, not insensitive as usually on windows)
- Consider adding a default route handler to match any differences

```
workbox.routing.registerRoute(
  ({ event }) => event.request.mode === 'navigate',
  async (event) => {
    var path = event.url.pathname.toLowerCase();
    if(!path.endsWith('index.html')) {
      if(!path.endsWith('/')) { path += '/'; }
      path += 'index.html';
    }
    return caches
      .match(workbox.precaching.getCacheKeyForURL(path))
      .then(response => { return response || fetch(path); })
      .catch(err => { return fetch(path); });
  }
);
```


Working with requests & responses

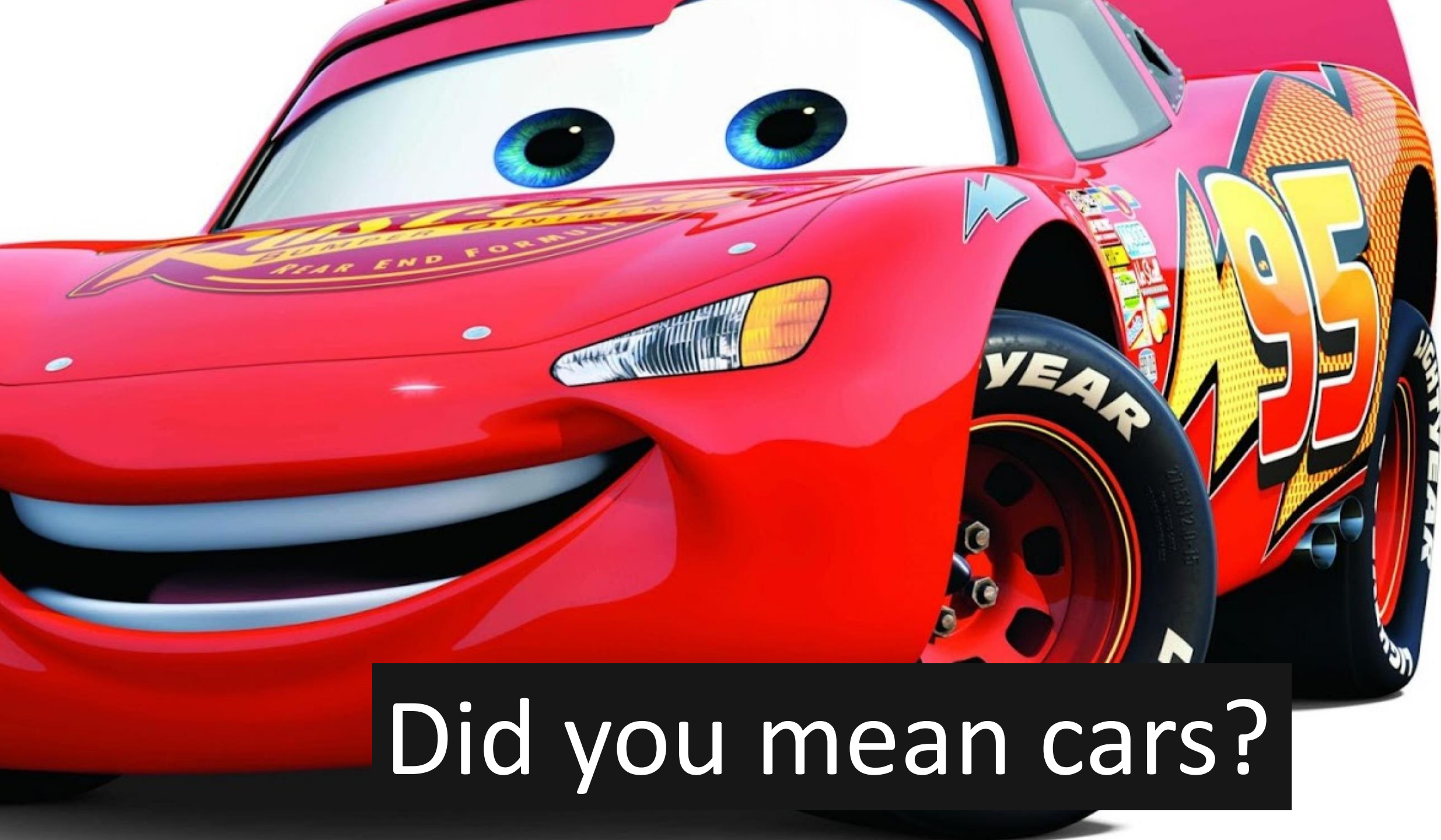
Workbox

Files

CORS

OAuth

Queueing
Requests



Did you mean cars?

Cross Origin Resource Sharing

Misconfigured CORS == No Caching

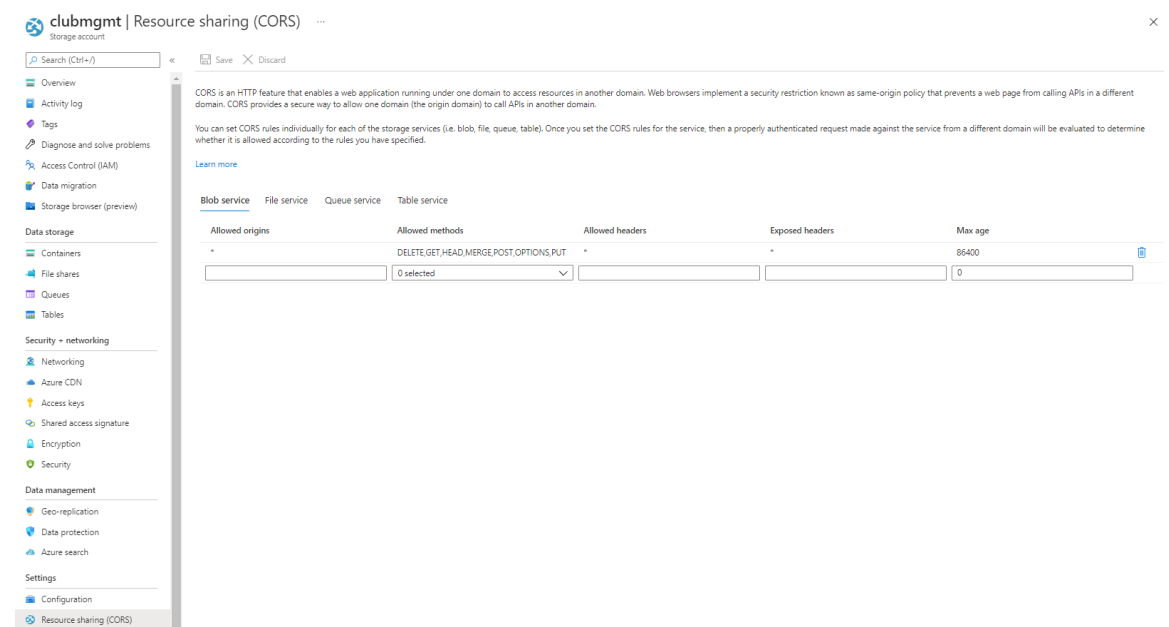
- Security feature which restricts scripts to interact with resources from other domains
- Opt-in on both client and server

Client

```
<script  
  src=https://clubmgmt.blob.core.windows.net/script.js  
  crossorigin="anonymous">  
</script>
```

- Must match
 - Origin
 - Method
 - Header

Server (blob storage)



clubmgmt | Resource sharing (CORS) ...

Storage account

Search (Ctrl+F) Save Discard

Overview
Activity log
Tags
Diagnose and solve problems
Access Control (IAM)
Data migration
Storage browser (preview)

Data storage
Containers
File shares
Queues
Tables

Security + networking
Networking
Azure CDN
Access keys
Shared access signature
Encryption
Security

Data management
Geo-replication
Data protection
Azure search

Settings
Configuration
Resource sharing (CORS)

CORS is an HTTP feature that enables a web application running under one domain to access resources in another domain. Web browsers implement a security restriction known as same-origin policy that prevents a web page from calling APIs in a different domain. CORS provides a secure way to allow one domain (the origin domain) to call APIs in another domain.

You can set CORS rules individually for each of the storage services (i.e. blob, file, queue, table). Once you set the CORS rules for the service, then a properly authenticated request made against the service from a different domain will be evaluated to determine whether it is allowed according to the rules you have specified.

Learn more

Blob service File service Queue service Table service

Allowed origins	Allowed methods	Allowed headers	Exposed headers	Max age
*	DELETE, GET, HEAD, MERGE, POST, OPTIONS, PUT	*	*	86400

CSS

Gotcha!

- **Requests originating from CSS have no cors headers!**

```
background: url(https://clubmgmt.blob.core.windows.net/image.jpg)
```

- Background image can't be cached
- Workaround, add prefetch with crossorigin attribute in html

```
<link  
  rel="prefetch"  
  href=https://clubmgmt.blob.core.windows.net/image.jpg  
  crossorigin="anonymous">
```

Force caching of opaque responses

For third party services

- No control on server side CORS configuration
- CORS violation = response code 0

```
workbox.routing.registerRoute(  
  new RegExp('https://www\\.gravatar\\.com/avatar/.*\\.(?:png|gif|jpg|svg).*'),  
  new workbox.strategies.CacheFirst({  
    cacheName: cacheName,  
    plugins: [  
      new workbox.cacheableResponse.CacheableResponsePlugin ({  
        statuses: [0, 200]  
      })  
    ]  
  })  
);
```



DANGER



May be caching a bad response and permanently break the file!

Working with requests & responses

Workbox

Files

CORS

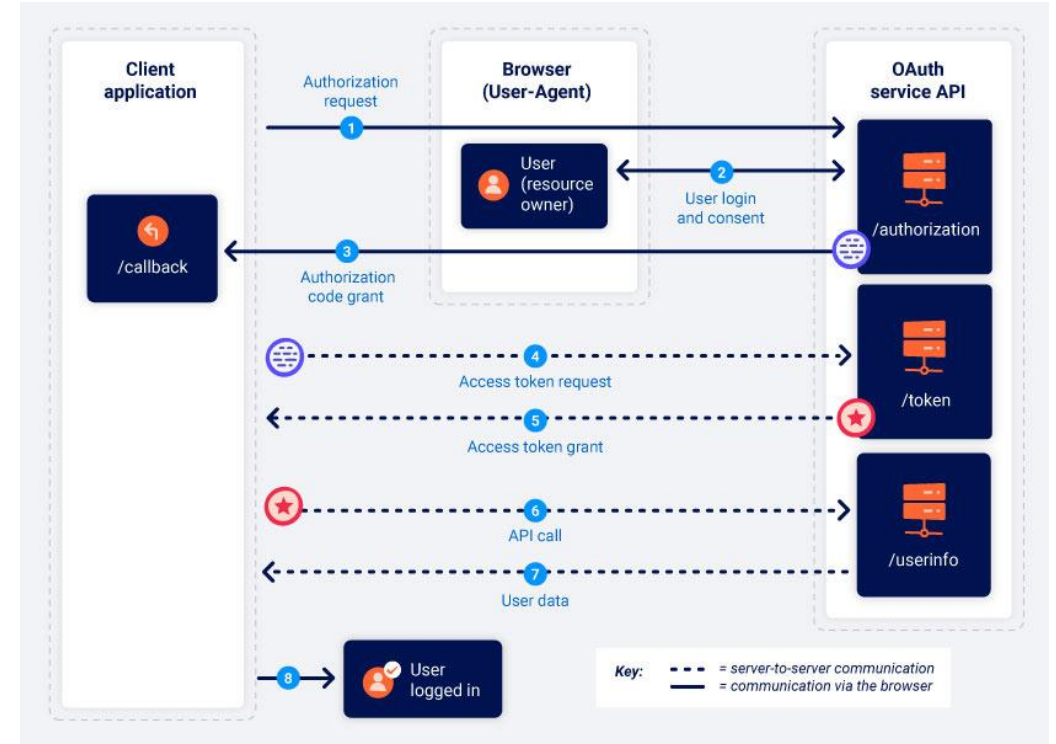
OAuth

Queueing
Requests

Oauth

Open standard authorization protocol

- Different grant types
- All rely on redirects and server calls
- Authorization code flow
 - Authorization request
 - Login & (optional) Consent
 - Identity token
 - Authorization code grant
 - Access token (expires)
 - User info
- Must be online while logging in!
- How to go, and stay, offline while access token expires?





There appears to be an `offline_access` scope?

Oauth offline_access scope

Not what it appears to be

- **Does not allow user access while user is offline!**
- Allows backend software to use refresh token on a user's behalf...





DANGER



Using `offline_access` in a frontend is dangerous!
(refresh token lifetime)



So how do I allow access while offline?



Disable silent renew while offline

Renew token when coming back online again

```
this.auth = new Oidc.UserManager(authConfig);

shell.topics.subscribe('offline', () => {
  this.auth.stopSilentRenew();
});

shell.topics.subscribe('online', () => {
  this.auth.startSilentRenew();

  if(this.tokenExpired()) this.auth.signinSilent();
});

If(shell.onLine)
{
  var identity = await this.auth.getUser();
  // see next slide...
}
```

Cache Identity & Access Token

Reuse cached identity & access token while offline

```
if(shell.onLine){
  var identity = await this.auth.getUser();
  if (identity) {
    this.identity = identity;
    this.cacheContext({ identity });
    this.topics.publish("authenticated", identity);
  }
  else {
    this.auth.clearStaleState(null).then(() => {
      this.auth.signinRedirect();
    });
  }
}
else{
  var ctx = this.getCachedContext();
  this.identity = ctx.identity;
  this.topics.publish("authenticated", this.identity);
}
```

Cache open id configuration file

Mind CORS!

- Proper CORS configuration on the server

```
app.UseCors(b => b.WithOrigins("https://*.clubmanagement.io")
    .AllowAnyHeader()
    .AllowAnyMethod());
```

- Allows caching openid configuration file

```
workbox.routing.registerRoute(
    new RegExp('^https://login.*.clubmanagement.io/.well-known/openid-configuration'),
    new workbox.strategies.StaleWhileRevalidate({
        cacheName: 'clubmgmt-authentication-cache'
    })
);
```


Working with requests & responses

Workbox

Files

CORS

OAuth

Queueing
Requests

Queue app insights events

Add failed requests to a queue

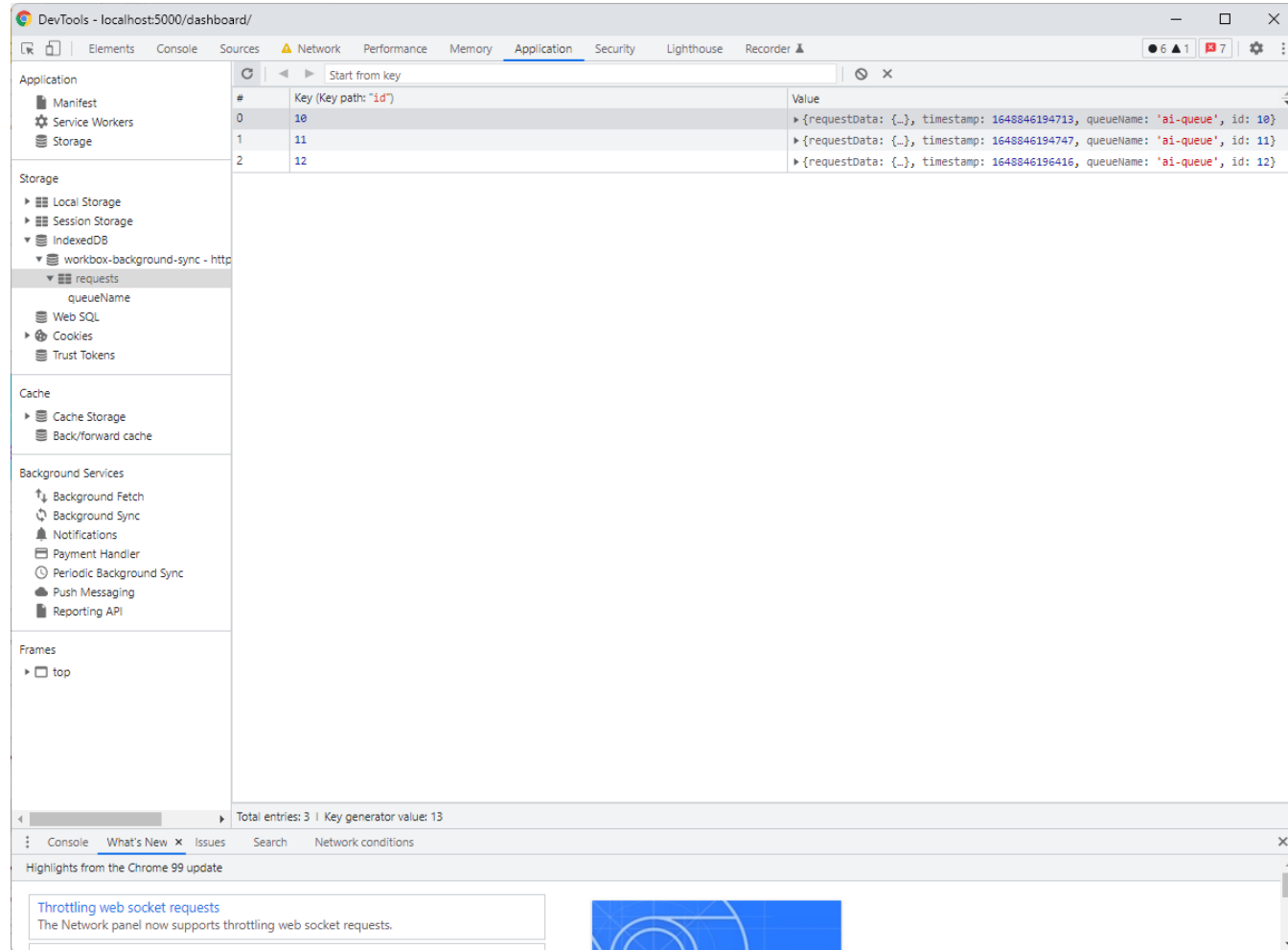
- Failed requests will be added to a queue
 - With automatic sync when coming back online
 - Using BackgroundSyncPlugin

```
const aiSync = new workbox.backgroundSync.BackgroundSyncPlugin('ai-queue', {
  maxRetentionTime: 24 * 60 // Retry for max of 24 Hours (specified in minutes)
});

workbox.routing.registerRoute(
  'https://dc.services.visualstudio.com/v2/track',
  new workbox.strategies.NetworkOnly({
    plugins: [aiSync]
  }),
  'POST'
);
```

Inspect queued requests

Application, storage, indexeddb



The screenshot shows the Chrome DevTools Application panel for a page at localhost:5000/dashboard/. The left sidebar is expanded to 'Storage' > 'IndexedDB' > 'workbox-background-sync - http' > 'requests'. The main pane displays a table of three queued requests:

#	Key (Key path: "id")	Value
0	10	{requestData: {...}, timestamp: 1648846194713, queueName: 'ai-queue', id: 10}
1	11	{requestData: {...}, timestamp: 1648846194747, queueName: 'ai-queue', id: 11}
2	12	{requestData: {...}, timestamp: 1648846196416, queueName: 'ai-queue', id: 12}

At the bottom of the panel, a status bar indicates 'Total entries: 3 | Key generator value: 13'. Below the DevTools window, a notification banner reads: 'Throttling web socket requests. The Network panel now supports throttling web socket requests.'

Cache app insights file

Opaque response caching required

```
workbox.routing.registerRoute(  
  new RegExp('https://az416426.vo.msecnd.net/scripts/b/ai.2.min.js'),  
  new workbox.strategies.CacheFirst({  
    cacheName: 'dish-monitoring-cache',  
    plugins: [  
      new workbox.cacheableResponse.CacheableResponsePlugin ({  
        statuses: [0, 200]  
      })  
    ]  
  })  
);
```

- We cannot control CORS config of app insights
 - Opaque response caching required
 - **Bad requests get cached as well!**

Requests with authorization headers

Bearer tokens may expire while in queue

```
const queue = new workbox.backgroundSync.Queue('bg-queue', {
  onSync: replayRequests,
  maxRetentionTime: 24 * 60 // Retry for max of 24 Hours (specified in minutes)
});
const bgSyncPlugin = {
  fetchDidFail: async ({ request }) => {
    await queue.pushRequest({ request });
  }
}
workbox.routing.registerRoute(
  /.+\./api\./.+/,
  new workbox.strategies.NetworkOnly({
    plugins: [bgSyncPlugin]
  }),
  'POST'
);
```

Replay when back online

Fix the auth headers before fetching

```
async function replayRequests(o){
  if(unableToSend()) return;
  while (entry = await o.queue.shiftRequest()) {
    try {
      if(unableToSend()){ // prevents infinite loop if connectivity drops while replaying
        await o.queue.unshiftRequest(entry); return;
      }
      var req = entry.request.clone();
      // fix headers with latest tokens
      req.headers.set('Authorization', "Bearer " + accessToken);
      await fetch(req);
    } catch (error) {
      await o.queue.unshiftRequest(entry);
    }
  }
}
```

Working with structured data

CQRS read

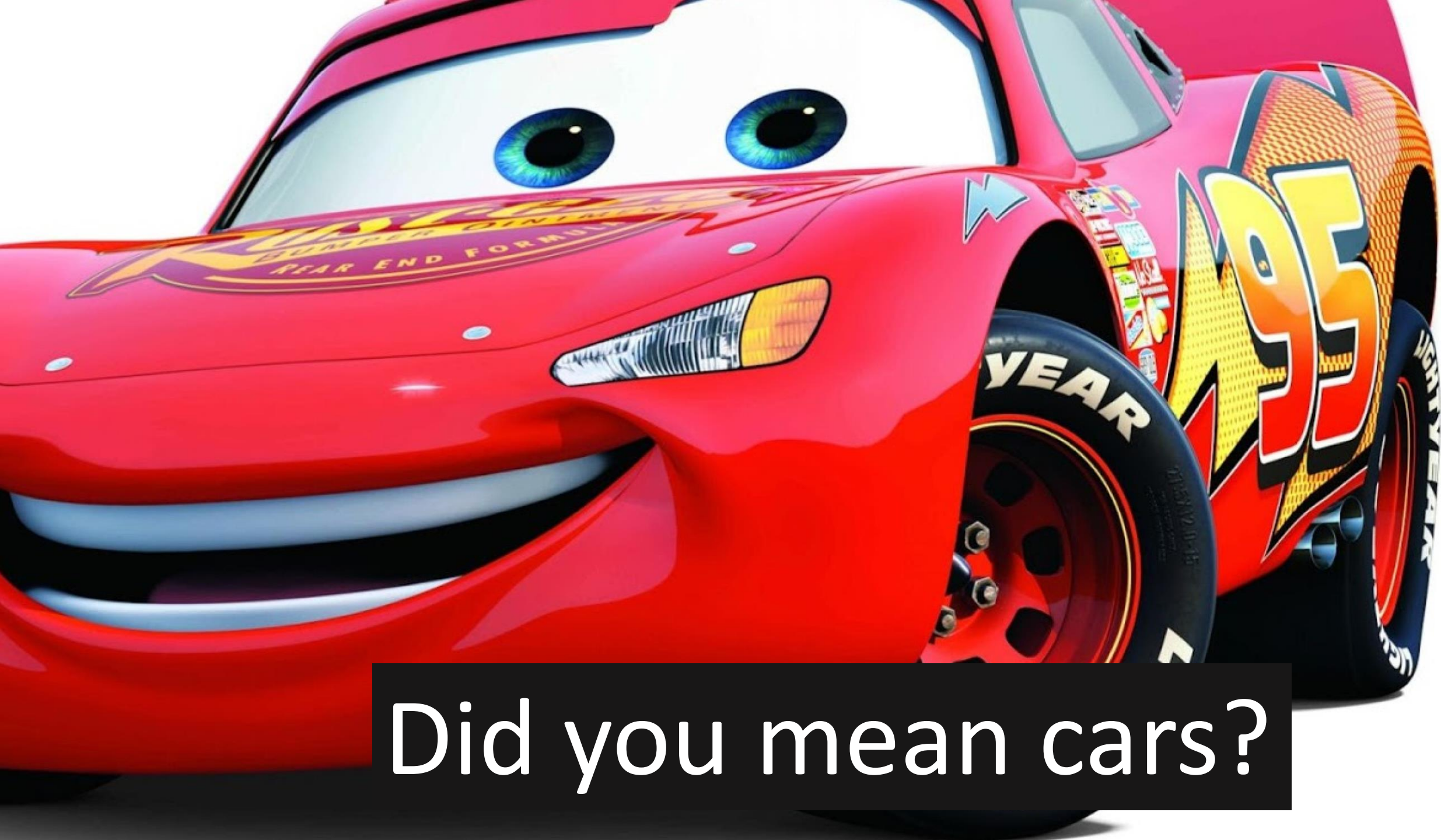
Inevitable Conflict

CRDTs

Conflict Handling

CQRS write

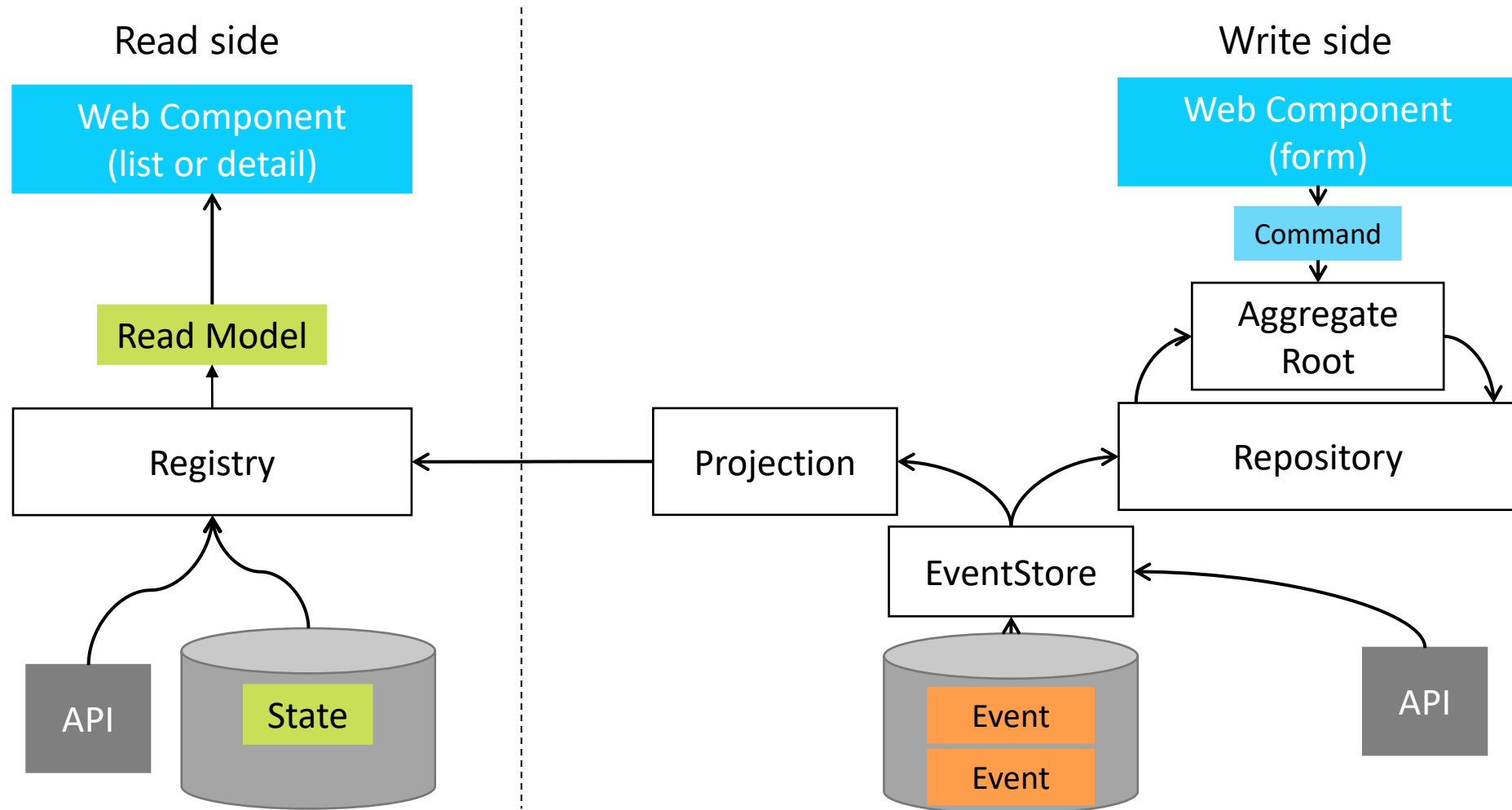
Synchronization



Did you mean cars?

Command Query Responsibility Segregation

Different design for reads and writes



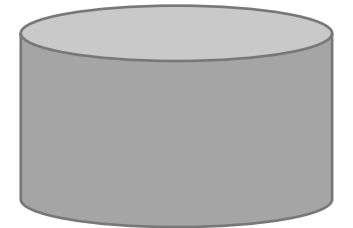
Read Side Example

Setting up indexeddb

```
class IndexedDBFactory{
  open(name, version, onupgrade){
    return new Promise((resolve, reject) => {
      var request = indexedDB.open(name, version);
      request.onerror = function(event) { reject(); };
      request.onsuccess = function(event) {
        var db = event.target.result;
        resolve(db);
      };
      request.onupgradeneeded = function(event) {
        var db = event.target.result;
        var tx = event.target.transaction;
        if(onupgrade) onupgrade(db, tx);
      };
    });
  }
}

var _db = await dbFactory.open(config.dbname, config.dbversion, (db, tx) => {
  new Registry(db).upgrade(tx);
});

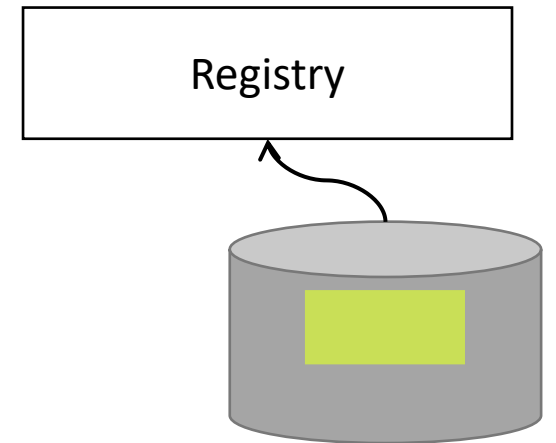
this.registry = new Registry(_db);
```



Read Side Example

Setting up tables and indexes

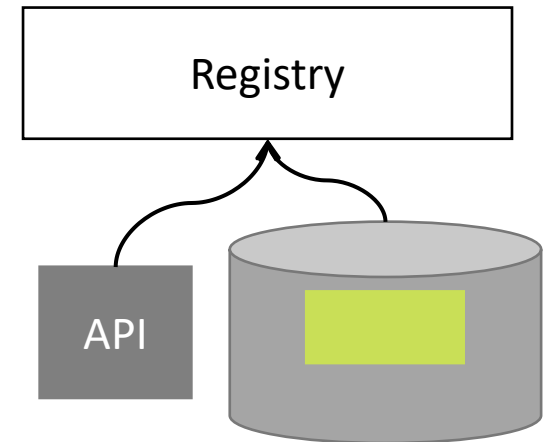
```
class Registry{  
  
  constructor(db,){ this.db = db; }  
  
  upgrade(tx){  
    var bookingsStore;  
    if (!this.db.objectStoreNames.contains("orderbookings")) {  
      bookingsStore= this.db.createObjectStore("orderbookings",{ keyPath: "id" });  
    }  
    else{  
      bookingsStore = tx.objectStore("orderbookings");  
    }  
  
    if(!bookingsStore.indexNames.contains("sale")) {  
      bookingsStore.createIndex("sale", "saleId");  
    }  
  
    if(!bookingsStore.indexNames.contains("seller")) {  
      bookingsStore.createIndex("seller", "sellerId");  
    }  
  }  
}
```



Read Side Example

Load records and handle exceptions when offline

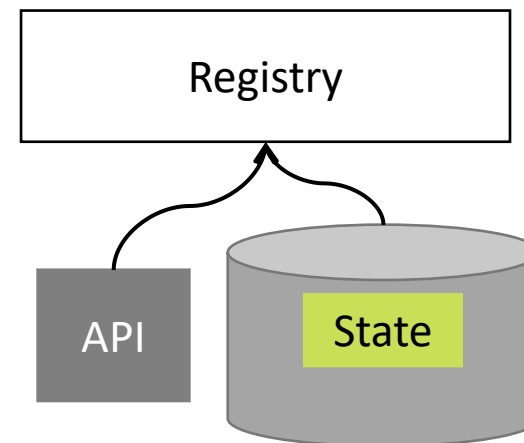
```
class Registry{  
  async loadOrderBookings(organizationId, saleId, requestId){  
  
    var uri = config.service + "/api/orderbookings/" + organizationId + "/" + saleId;  
    let bookings;  
    try  
    {  
      let request = await fetch(uri, {  
        method: "GET",  
        mode: 'cors',  
        headers: {  
          "Content-Type": "application/json",  
          "Authorization": "Bearer " + authentication.getAccessToken()  
        }  
      });  
  
      bookings = await request.json();  
    }  
    catch(ex) { if (shell.onLine) { throw ex; } }  
  
    if(bookings) await this.persistOrderBookings(bookings, true);  
  
    app.topics.publish("orderbookings.refreshed", true);  
  }  
}
```



Read Side Example

Store records in table

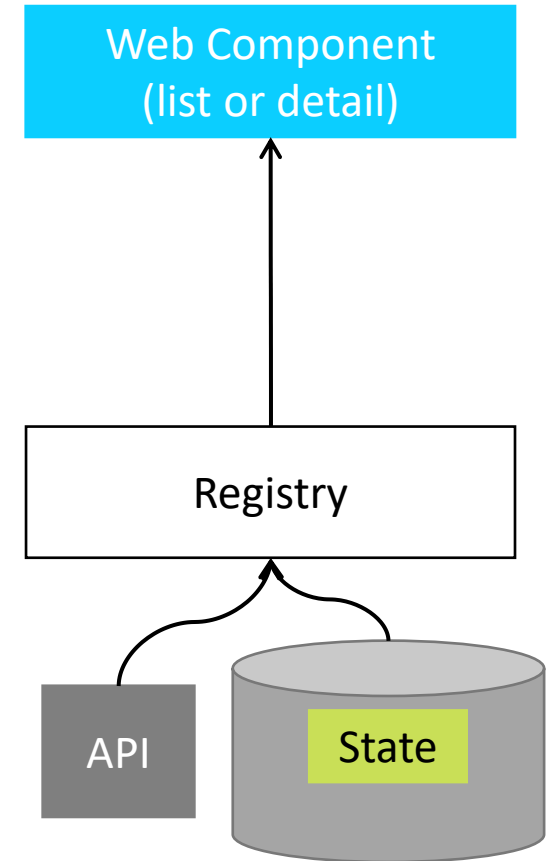
```
class Registry{  
  
  async persistOrderBookings(bookings, restoring){  
    return new Promise((resolve, reject) => {  
      var tx = this.db.transaction("orderbookings", "readwrite");  
      var store = tx.objectStore("orderbookings");  
  
      tx.onerror = () => {  
        app.topics.publish("orderbookings.persistenceFailed", restoring);  
        reject();  
      }  
  
      tx.oncomplete = () => {  
        app.topics.publish("orderbookings.persisted", restoring);  
        resolve();  
      };  
  
      for (const booking of bookings) {  
        store.put(booking);  
      }  
    });  
  }  
}
```



Read Side Example

Web component

```
class OrderBookingTotal extends HTMLElement {  
  
  constructor() {  
    super();  
  }  
  
  async connectedCallback() {  
  
    var booking = await app.registry.getOrderBooking(this.orderId);  
    if(!booking) return;  
  
    var total = 0;  
    var currency = "";  
  
    for(var line of booking.orderLines){  
      currency = line.orderedItem.price.currency;  
      total += line.quantity * line.orderedItem.price.value;  
    }  
  
    var summary = currency + Math.round((total + Number.EPSILON) * 100) / 100;  
  
    if(this.innerHTML != summary){  
      this.innerHTML = summary;  
    }  
  }  
}
```



Working with structured data

CQRS read

Inevitable Conflict

CRDTs

Conflict Handling

CQRS write

Synchronization

Conflict is inevitable

While making changes offline

- The longer you work offline.
- The higher the chance someone else modified the same data you did.
- Leading to conflicts when returning online.
- 2 ways to deal with this:
 1. Learning to model your data differently
 - Model process steps instead of entities
 - Outside the scope of this talk
 - All our aggregates are wrong (Mauro Servienti)
 2. Restricting data structures to CRDTs (conflict free replicated data types)
 - Let's explore...

Working with structured data

CQRS read

Inevitable Conflict

CRDTs

Conflict Handling

CQRS write

Synchronization

CRDTs

Conflict-free replicated data types

A data structure which can be replicated across multiple computers in a network, where the replicas can be updated independently and concurrently without coordination between the replicas, and where it is always mathematically possible to resolve inconsistencies that might come up

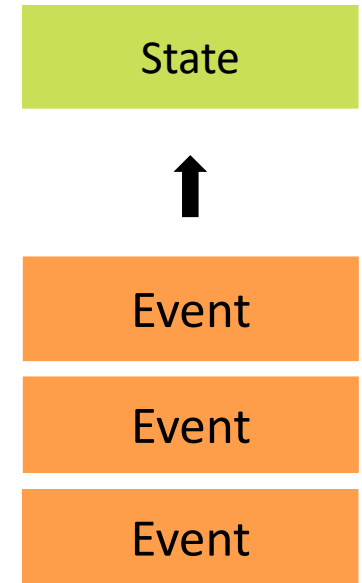
- Operation-based CRDTs: operations must be commutative
- State-based CRDTs: state changes must be commutative, associative and idempotent
- Known CRDTs
 - Grow-only counter
 - Positive-Negative counter
 - **Grow-only set**
 - Two-phase set
 - Last-Write-Wins-Element-Set
 - Observed-Remove-Set
 - Sequence

Event Sourcing

Persistence Mechanism

Event Sourcing is a persistence mechanism where each state transition for a given entity is represented as a domain event that gets persisted to an event database.

- Event store => Grow-only set (operation based CRDT)
- Under condition that order doesn't matter => commutative
- Guarantee that any conflict can be resolved



Working with structured data

CQRS read

Inevitable Conflict

CRDTs

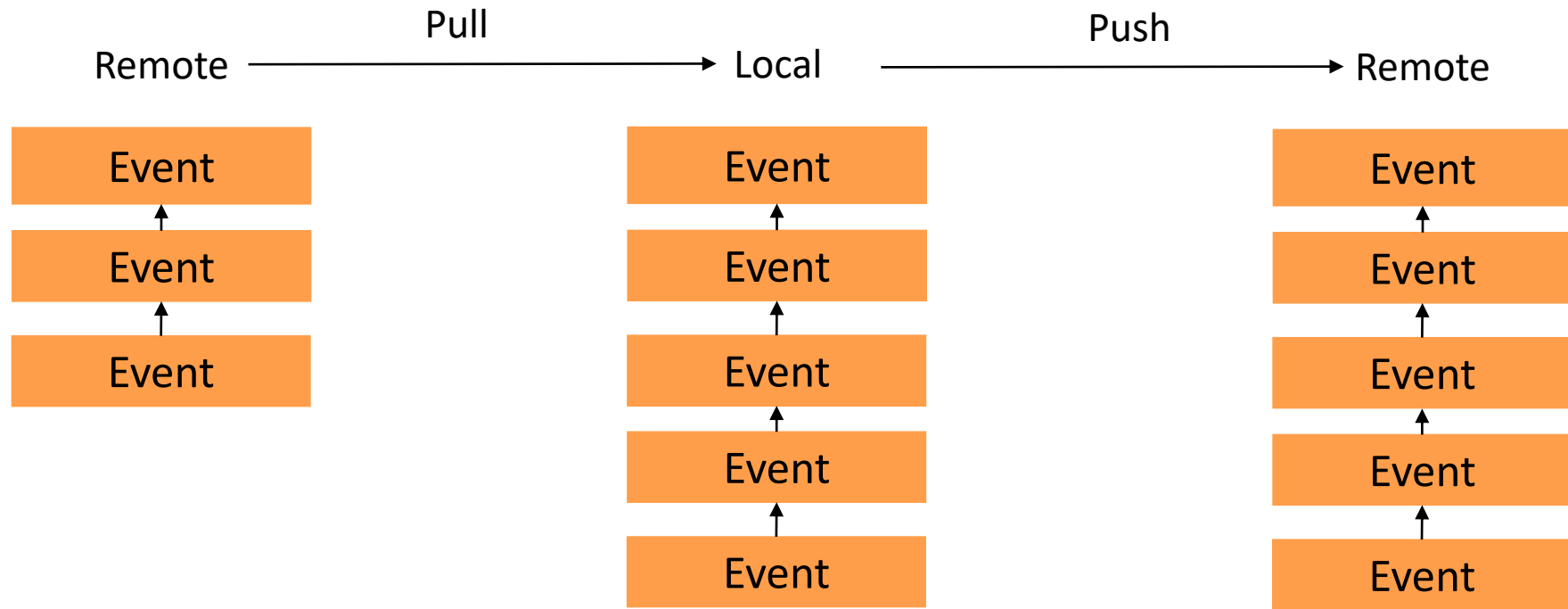
Conflict Handling

CQRS write

Synchronization

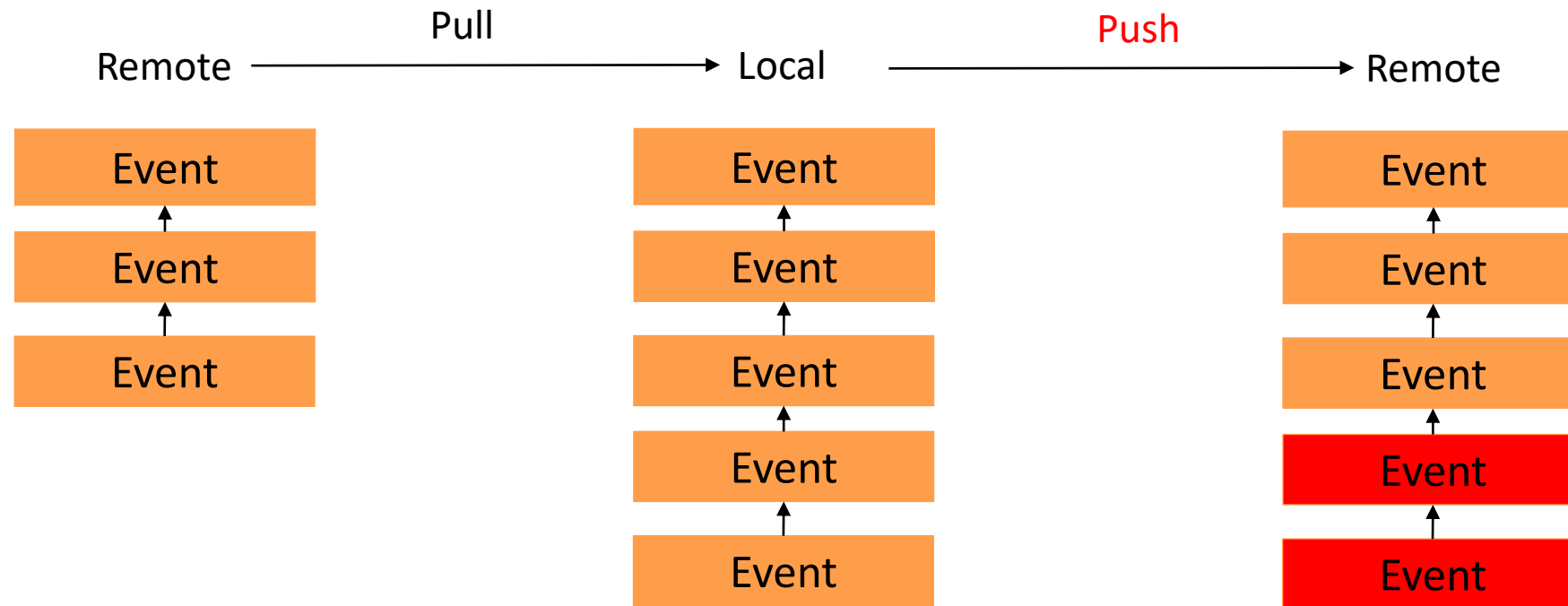
Conflict Resolution Inspired By Git

Pull & Push



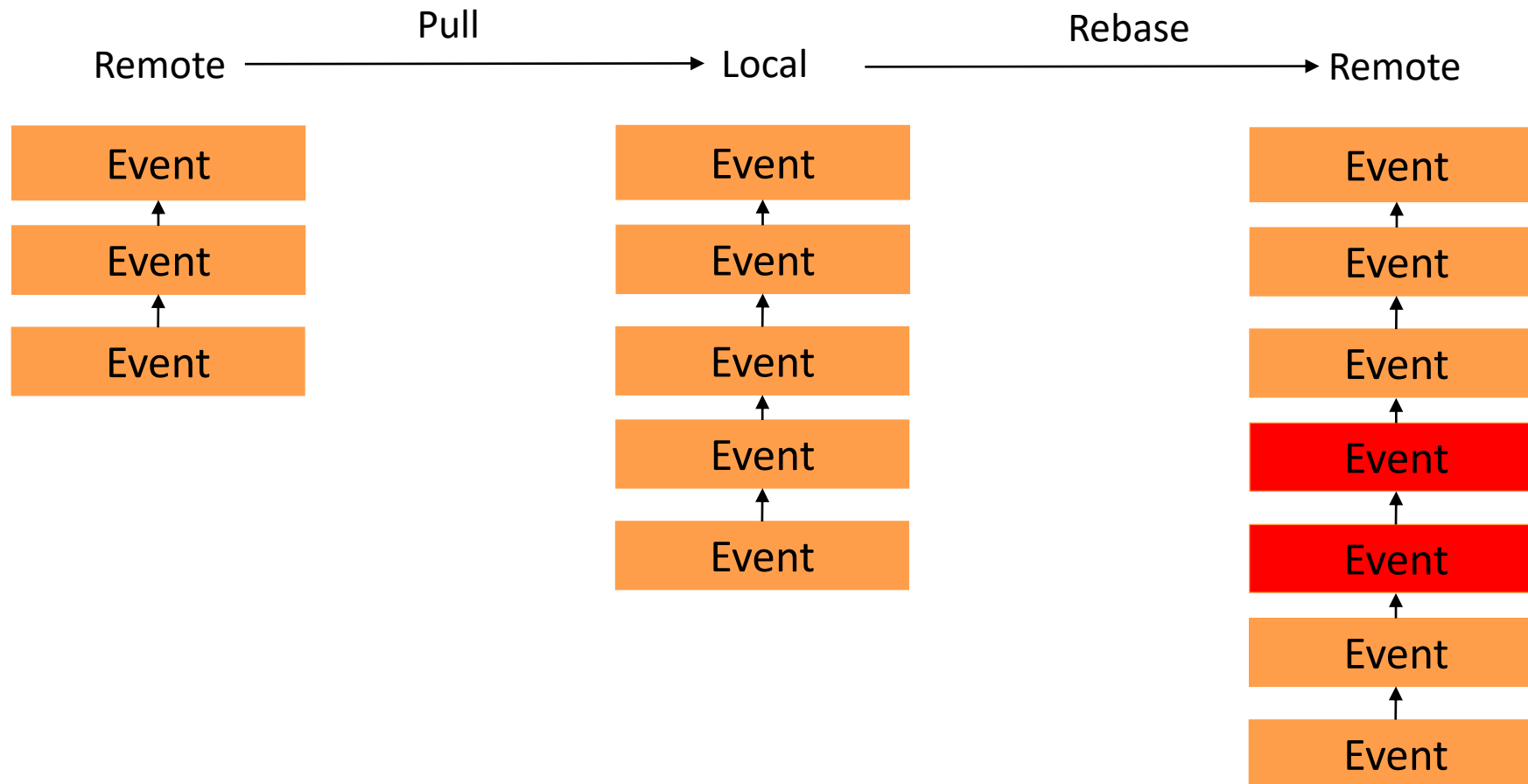
Conflict Resolution

Push will be rejected when conflicts



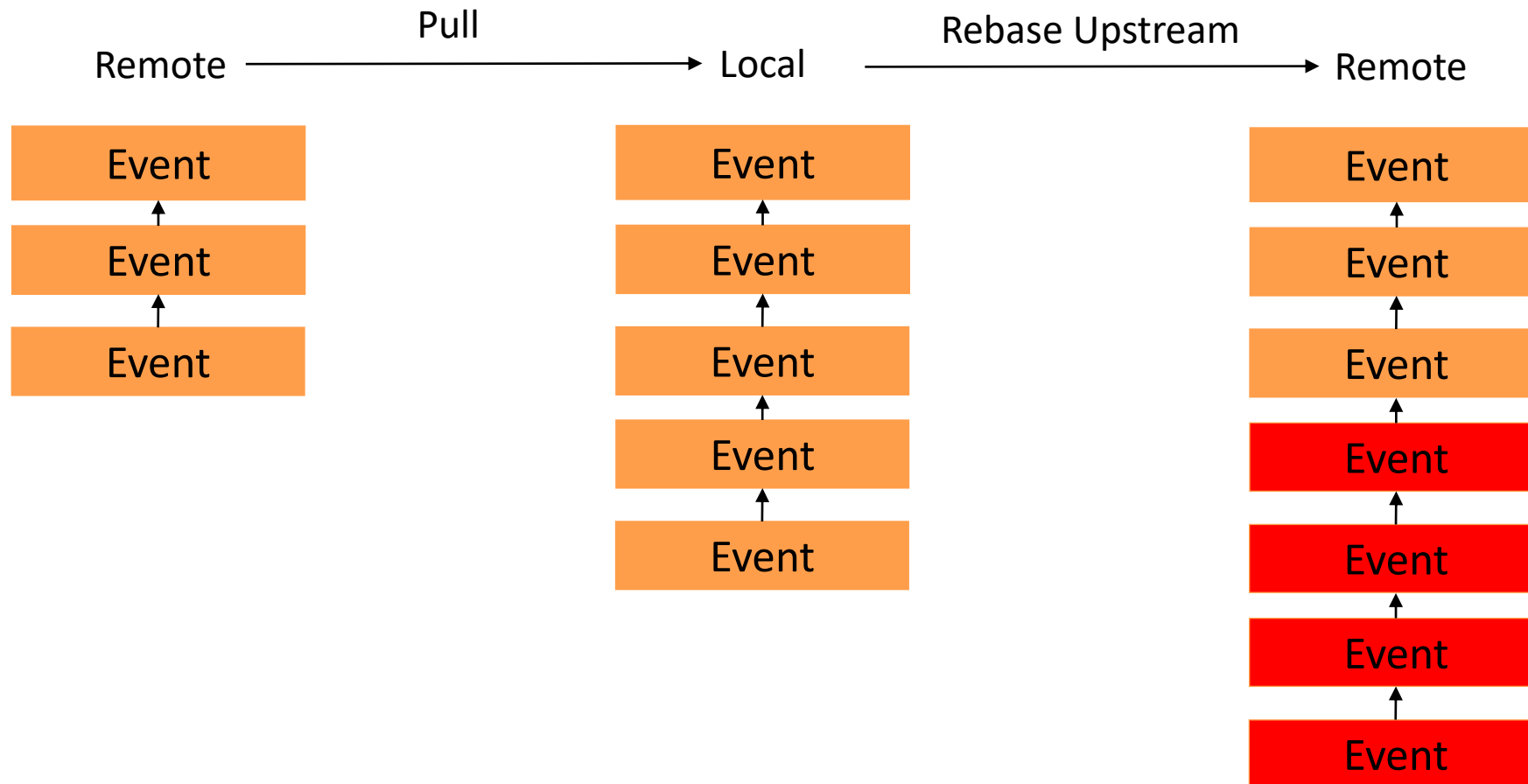
Conflict Resolution

Commutative = Order does not matter => Rebase



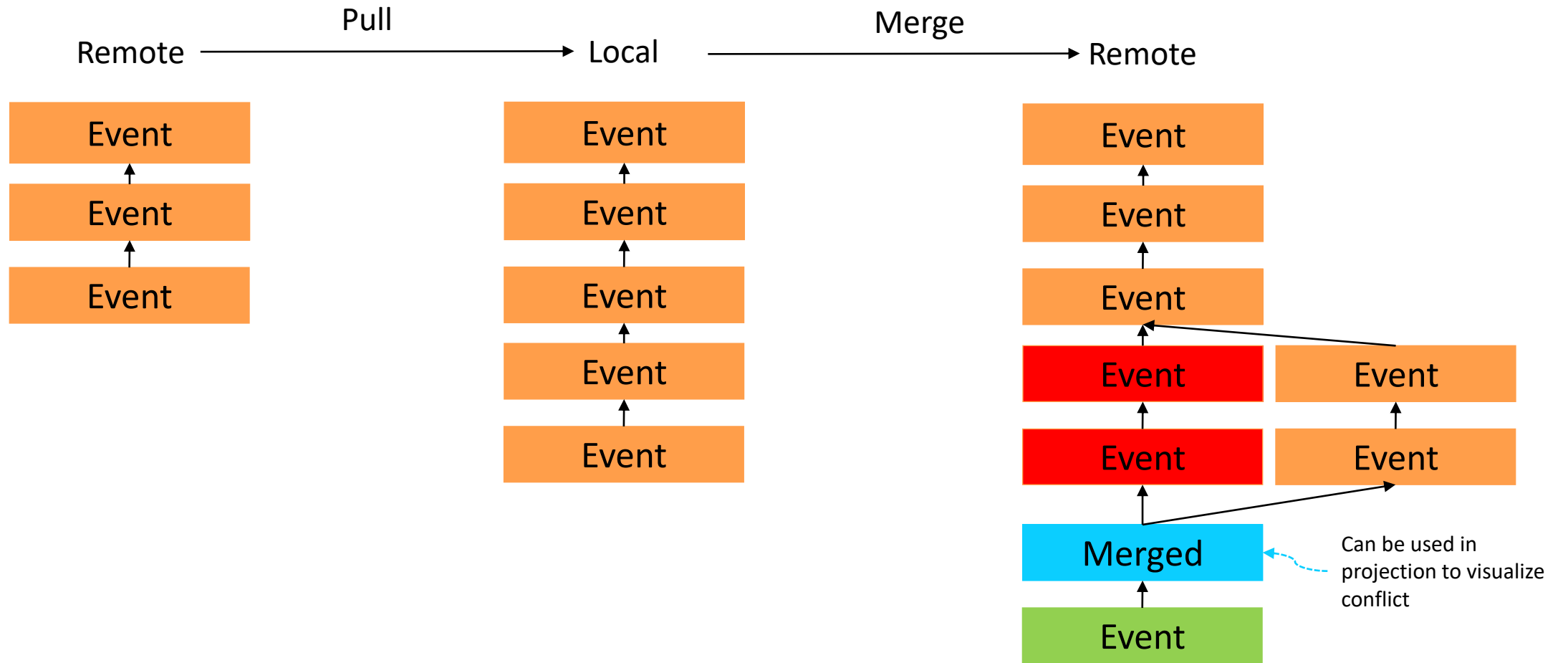
Conflict Resolution

Commutative = Order does not matter => OR Rebase Upstream



Conflict Resolution

What if order does matter? => Merge => Fix with future event



Working with structured data

CQRS read

Inevitable Conflict

CRDTs

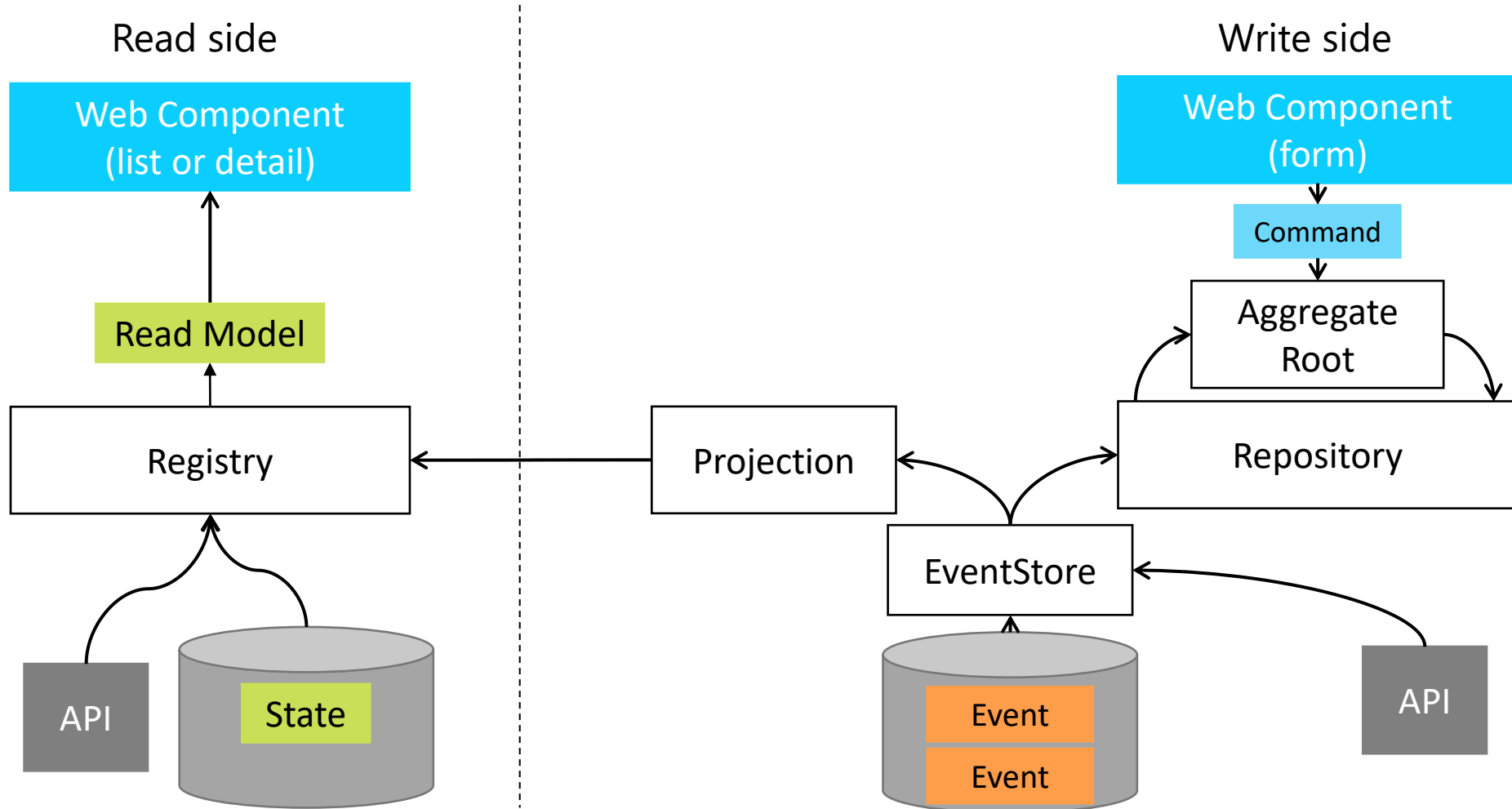
Conflict Handling

CQRS write

Synchronization

CQRS Write side

Working offline



Write Side Example

Event Store tables

```
class EventStore{

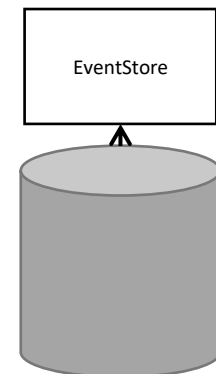
  constructor(db, table){
    this.db = db;
    this.table = table    }

  upgrade(tx){
    var eventStore;
    if (!this.db.objectStoreNames.contains(this.table)) {
      eventStore = this.db.createObjectStore(this.table, { keyPath: "event.id" });
    }
    else{
      eventStore = tx.objectStore(this.table);
    }

    if(!eventStore.indexNames.contains('sourceId')) {
      eventStore.createIndex("sourceId", "event.subject");
    }

    if(!eventStore.indexNames.contains('tenantId')) {
      eventStore.createIndex("tenantId", "event.cmtenantid");
    }

    if(!eventStore.indexNames.contains('when')) {
      eventStore.createIndex("when", "event.time");
    }
  }
}
```

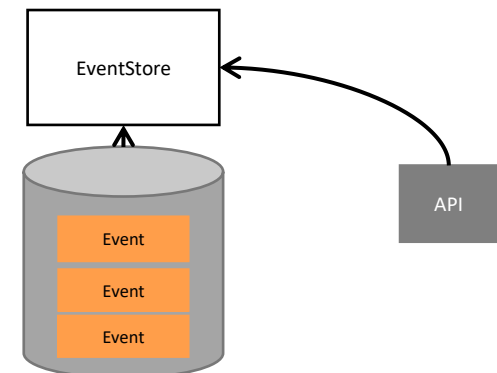


Write Side Example

Event Store load events

```
class OrderBookingsEventStore{

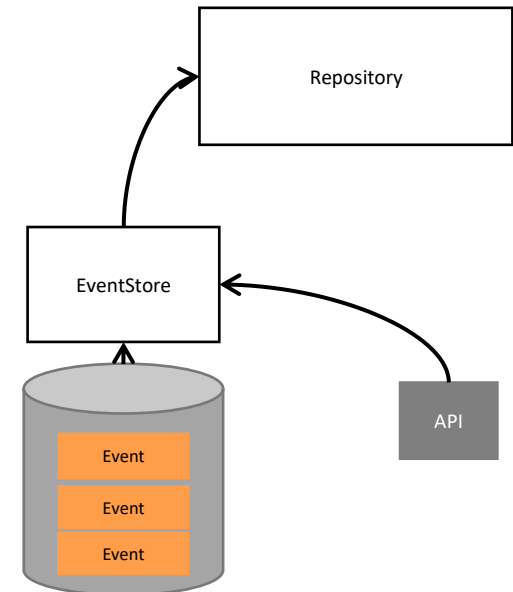
  constructor(db){
    this.uri = config.service + "/api/orderbookings/";
    this.eventStore = new EventStore(db, 'orderbookings');
  }
  async populate(sellerId, requestId){
    var uri = this.uri + "sellers/" + sellerId + "/events/pull";
    try
    {
      var response = await fetch(uri, {
        method: "GET",
        mode: 'cors',
        headers: {
          "Content-Type": "application/json",
          "Authorization": "Bearer " + authentication.getAccessToken()
        }
      });
      if(response.status == 200){
        var events = await response.json();
        await this.eventStore.persistEvents(events, true);
      }
    }
    catch(ex) { if (shell.onLine) { throw ex; } }
    app.topics.publish("orderbookings.refreshed", sellerId);
  }
}
```



Write Side Example

Repository

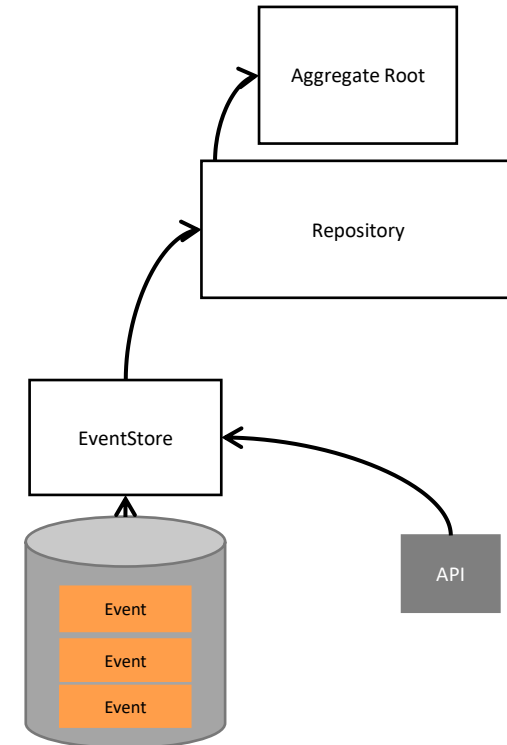
```
class OrderBookingRepository{  
  
    constructor(eventStore){  
        this.eventStore = eventStore;  
    }  
  
    async populate(sellerId){  
        await this.eventStore.populate(sellerId);  
    }  
  
    async getBooking(bookingId){  
        var events = await this.eventStore.get(bookingId);  
        var booking = new OrderBooking(bookingId);  
        booking.restoreFrom(events);  
        return booking;  
    }  
}
```



Write Side Example

Restoring an Aggregate Root from past events

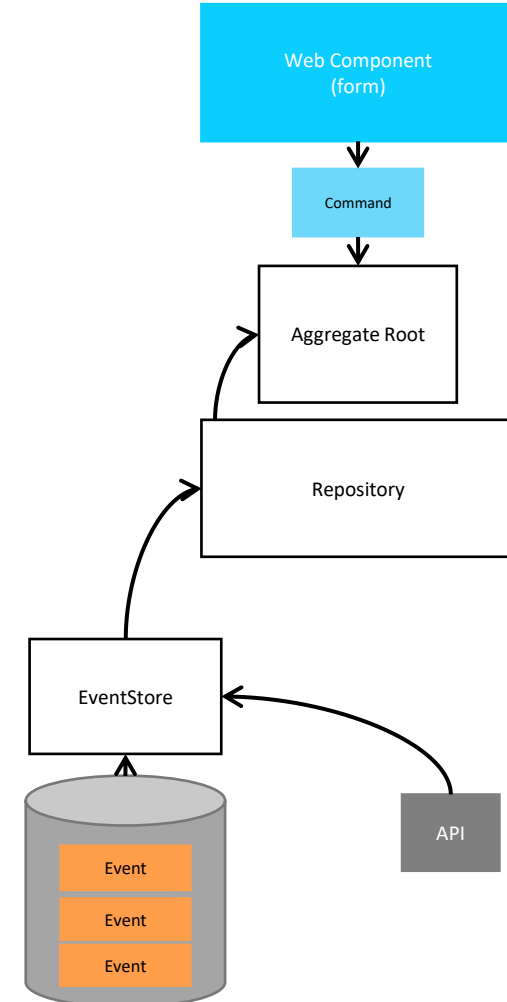
```
class OrderBooking extends EventSourced{  
  
  constructor(id){  
    super(id,  
          "https://contracts.clubmanagement.io/orderbooking/",  
          "io.clubmanagement.contracts.orderbooking");  
  }  
  
  applySalesOrderBooked(evt){  
    this.id = evt.data.salesOrder.id;  
    this._order = evt.data.salesOrder;  
  };  
  
  applySalesOrderCancelled(evt){};  
  
  applySalesOrderModified(evt){};  
  
  applySalesOrderCommented(evt){};  
  
  applyDiscountGranted(evt){};  
  
  applyDiscountRevoked(evt){}  
  
  applySalesOrderDeliveryExpectationsChanged(evt){ }  
}
```



Write Side Example

Web Component

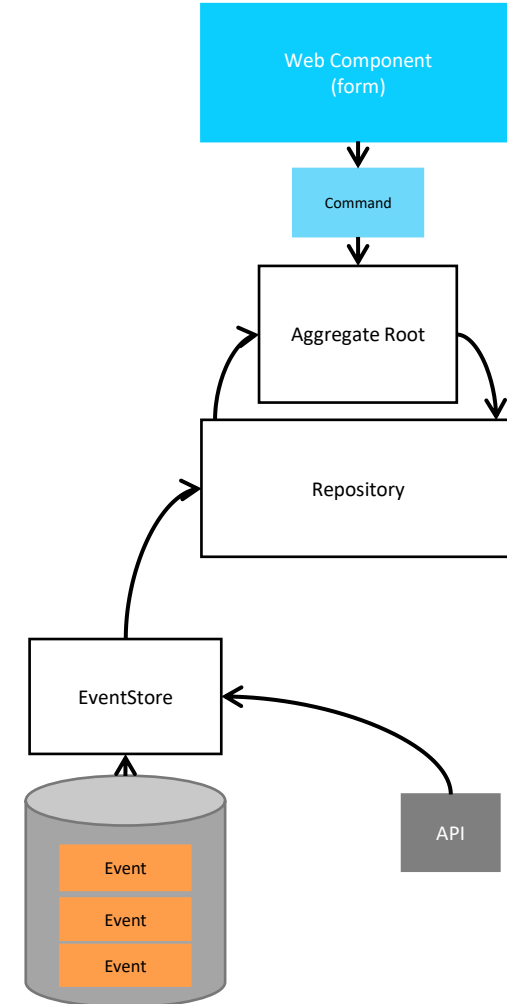
```
class BookOrder extends HTMLElement {  
  
  async connectedCallback() {  
  
    var form = this.querySelector('form');  
    form.addEventListener('submit', async (event) => {  
      event.preventDefault();  
      var bookingId = guid.generate();  
      var buyer = { name : form.querySelector('#name').value };  
      var orderlines = this.orderlineSelector.getOrderLines();  
      var deliveryExpectations = this.deliveryOptionsSelector.getDeliveryExpectations();  
  
      var booking = await app.repository.getBooking(bookingId);  
  
      booking.book(this.saleId, this.sellerId, buyer, orderlines, deliveryExpectations);  
  
      await app.repository.persistBooking(booking);  
      await app.repository.flush(this.sellerId);  
  
      var projected = await app.projection.projectBooking(bookingId);  
      await orderbooking.registry.persistOrderBooking(projected);  
    });  
  }  
}
```



Write Side Example

Aggregate Root

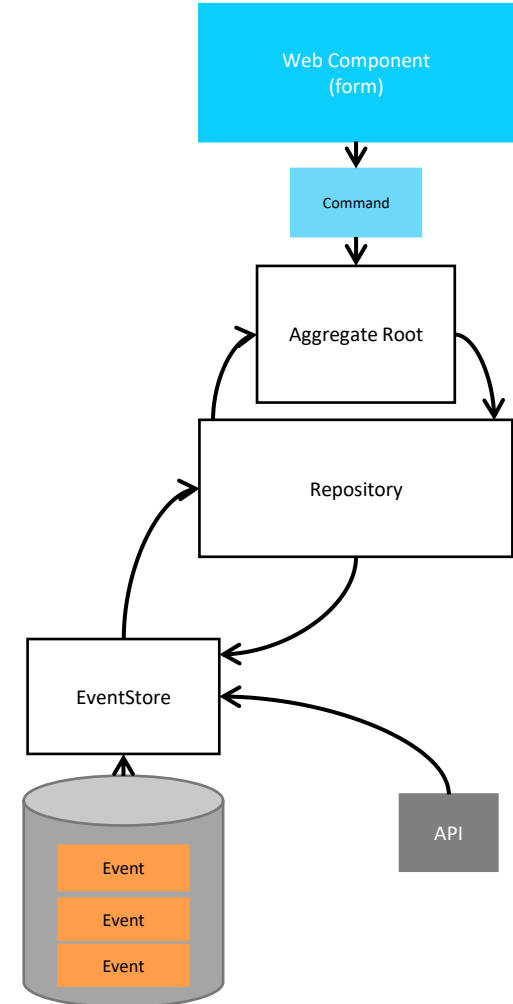
```
class OrderBooking extends EventSourced{  
  
  book(saleId, sellerId, buyer, orderLines, statusUpdateRequested, deliveryExpectations){  
  
    if (this._order != null) return;  
  
    super.emit({  
      tenantId: sellerId,  
      context: {  
        id: this.id,  
        what: "SalesOrderBooked",  
        when: new Date(),  
        who: authentication.identity.profile.id  
      },  
      salesOrder: {  
        id: this.id,  
        buyer: buyer,  
        sellerId: sellerId,  
        saleId: saleId,  
        orderLines: orderLines,  
        statusUpdatesRequested: statusUpdateRequested,  
        deliveryExpectations: deliveryExpectations  
      }  
    });  
  }  
}
```



Write Side Example

Repository

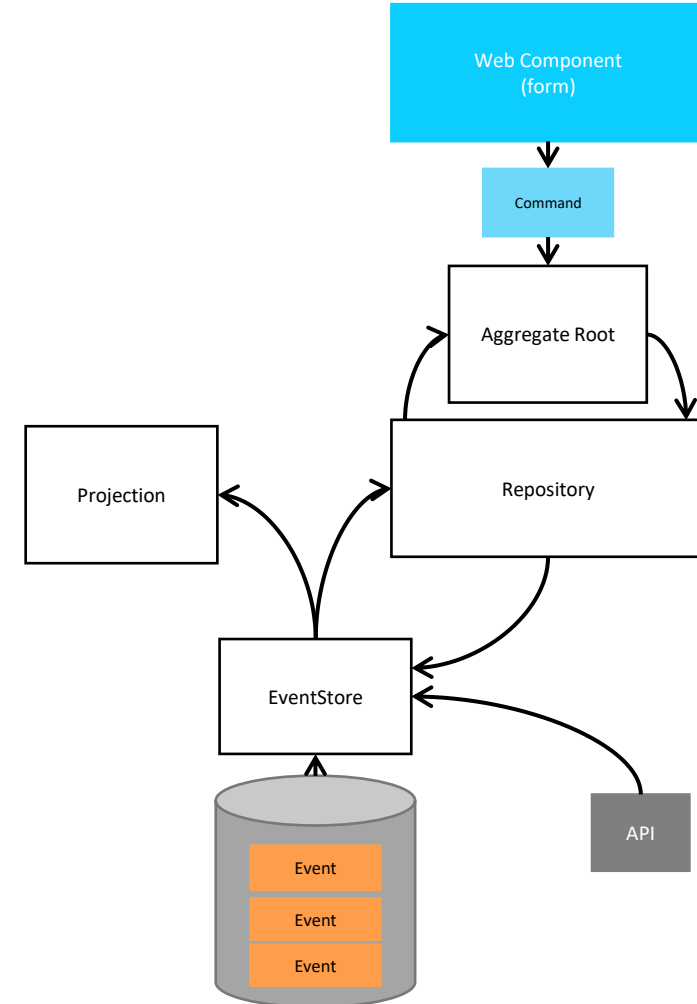
```
class OrderBookingRepository{  
  async persistBooking(booking){  
    var events = booking.commit();  
    for (const e of events) {  
      await this.eventStore.persistEvent(e, false);  
  
      app.topics.publish(e.data.context.what, e);  
    }  
    app.topics.publish("booking.persisted", booking.id);  
  }  
}
```



Write Side Example

Projection

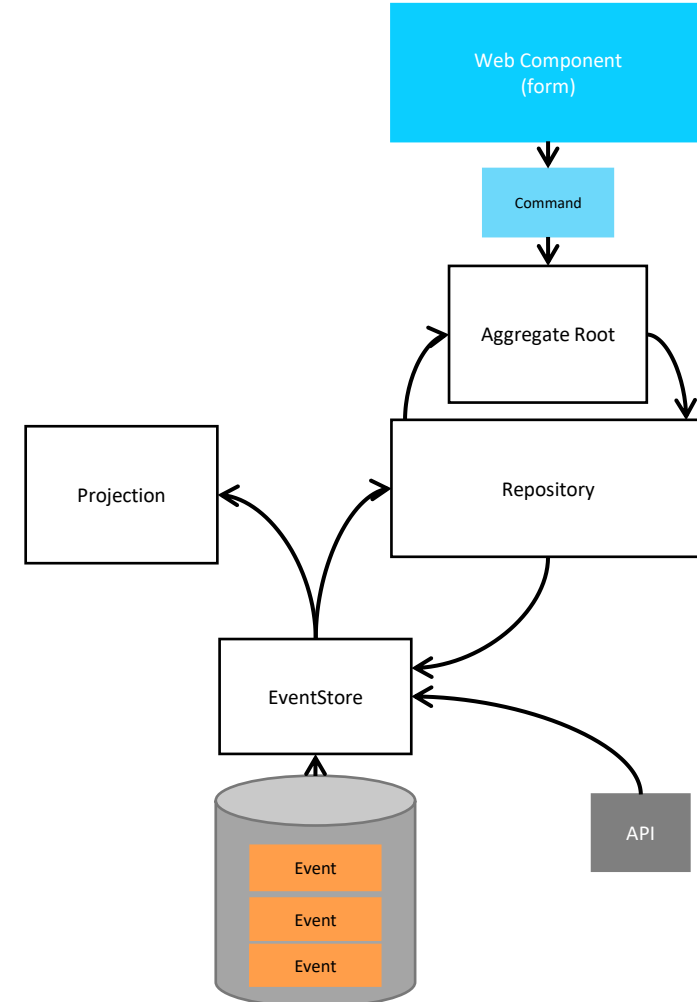
```
class OrderBookingsProjection{  
  
  constructor(eventStore, db){  
    this.eventStore = eventStore;  
  }  
  
  async populate(sellerId){  
    await this.eventStore.populate(sellerId);  
  }  
  
  async projectBooking(bookingId){  
    var events = await this.eventStore.get(bookingId);  
    var projection = new OrderBookingProjection();  
    var bookings = projection.project(events);  
    return bookings[0];  
  }  
  
  async projectBookings(saleId){  
    var projection = new OrderBookingProjection();  
    var events = await this.eventStore.all();  
    var bookings = projection.project(events);  
    return bookings.filter(r => r.saleId == saleId);  
  }  
  
}
```



Write Side Example

Projection

```
class OrderBookingProjection extends Projection{  
  
  projectSalesOrderBooked(order, evt){  
    order.id = evt.data.salesOrder.id;  
    order.referenceNumber = evt.data.referenceNumber;  
    order.saleId = evt.data.salesOrder.saleId;  
    order.sellerId = evt.data.salesOrder.sellerId;  
    order.buyer = evt.data.salesOrder.buyer;  
    order.statusUpdatesRequested = evt.data.salesOrder.statusUpdatesRequested;  
    order.orderLines = evt.data.salesOrder.orderLines;  
    order.orderDate = evt.data.context.when;  
    order.deliveryExpectations = evt.data.salesOrder.deliveryExpectation;  
    order.comment = evt.data.comment  
  };  
  
  projectSalesOrderCommented(order, evt){  
    order.comment = evt.data.comment;  
  };  
  
  projectSalesOrderCancelled(order, evt){  
    order.cancelled = true;  
  };  
}
```



Working with structured data

CQRS read

Inevitable Conflict

CRDTs

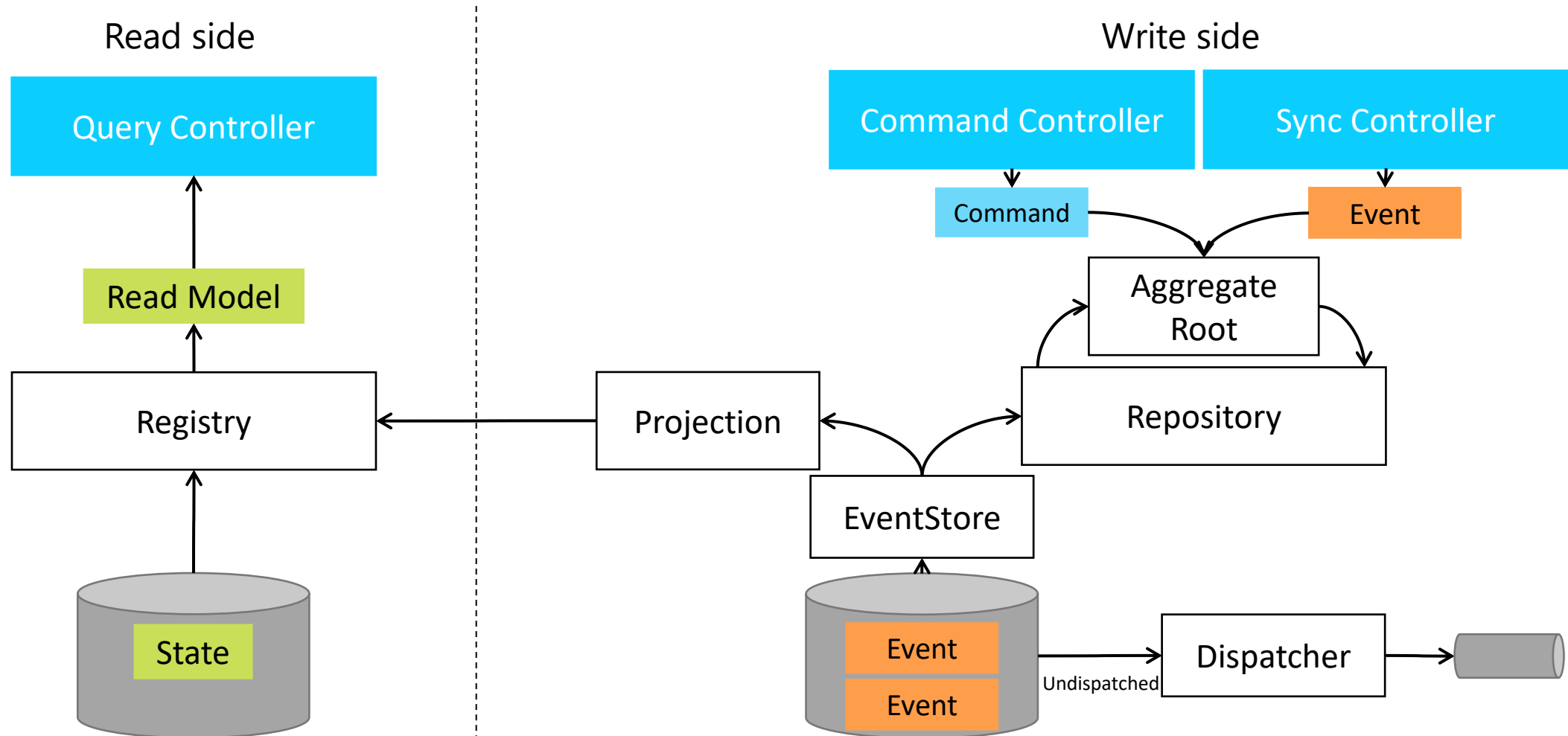
Conflict Handling

CQRS write

Synchronization

API Architecture

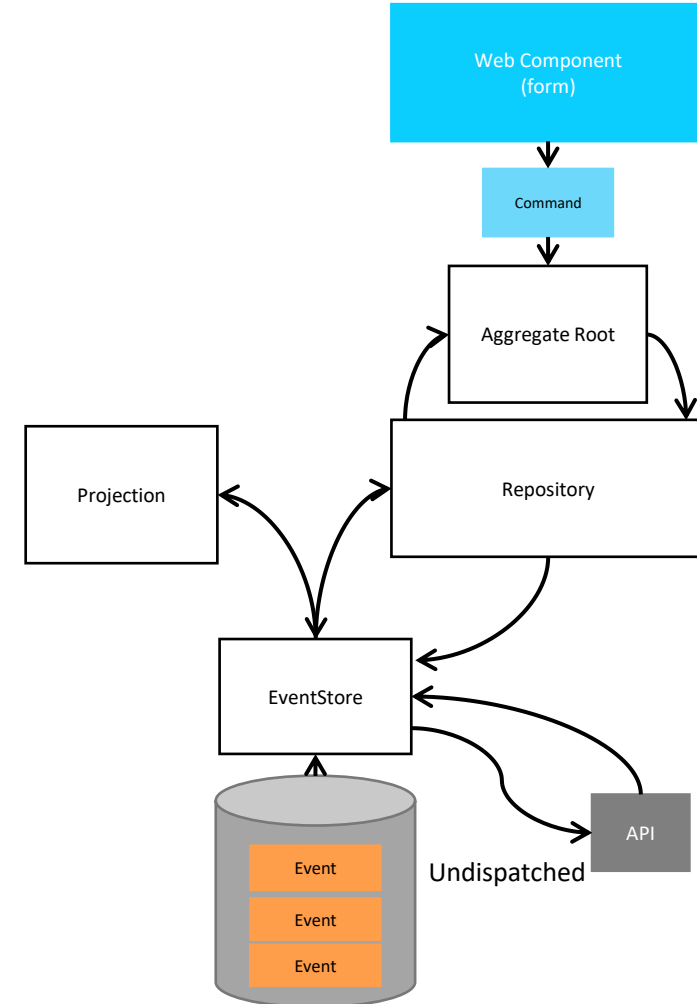
Same as the client-side architecture + sync, but written in C#



Synchronization

Frontend side

```
class OrderBookingsEventStore{
  async flush(tenantId){
    var entities = await this.eventStore.undispatched(tenantId);
    var entityIds = Object.keys(entities);
    if(entityIds && entityIds.length > 0){
      for (const id of entityIds) {
        var undispatched = entities[id];
        var uri = this.uri + id + "/events/rebase"; // or merge
        try
        {
          var response = await fetch(uri, {
            method: "POST",
            mode: 'cors',
            headers: {
              "Content-Type": "application/cloudevents+json",
              "Authorization": "Bearer " + authentication.getAccessToken()
            },
            body: JSON.stringify(undispatched)
          });
          if(response.status == 200){
            for (const e of undispatched) {
              await this.eventStore.persistEvent(e, true);
            }
          }
        }
      }
    }
  }
  catch(ex) { if (shell.onLine) { throw ex; } }
}
```



Synchronization

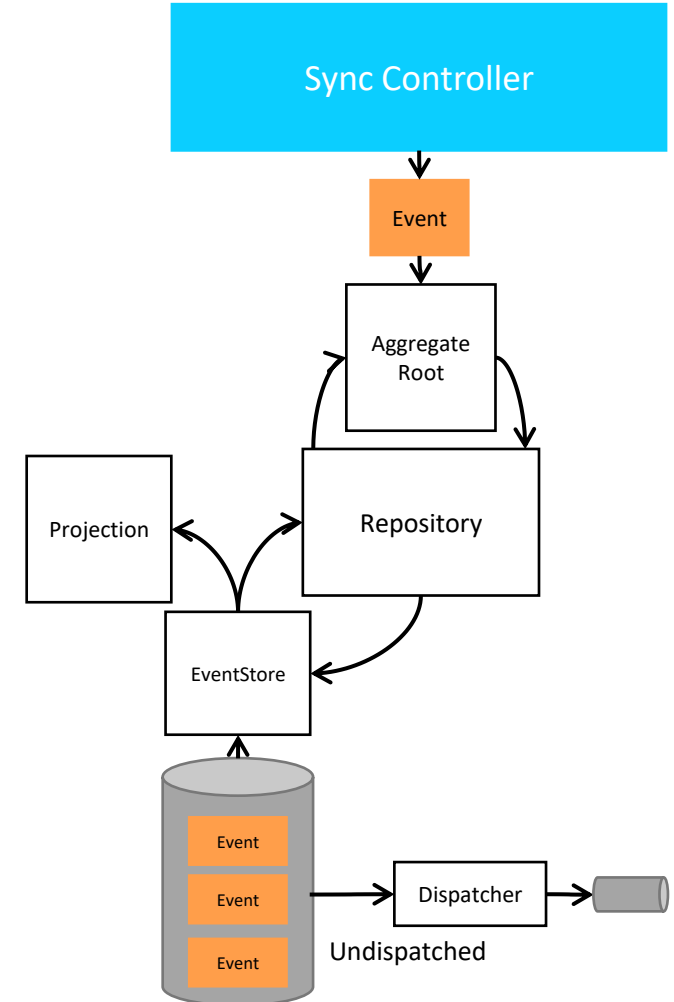
API side

```
[ApiController]
[Route("api/orderbookings")]
public class SynchronizationController : ControllerBase
{
    private readonly IEventSourcedRepository _repository;

    [HttpPost()]
    [Route("{id}/events/rebase")]
    [Authorize]
    public async Task Rebase([FromRoute] string id)
    {
        var events = await Request.Body.DecodeAsSourcedEventsAsync();
        if (events == null)
        {
            Response.StatusCode = StatusCodes.Status400BadRequest;
            return;
        }

        var aggregate = await _repository.Get<OrderBooking>(id);
        var rebased = aggregate.Rebase(events);
        await _repository.Flush();

        var encoded = rebased.EncodeAsCloudEvents(out var contentType);
        Response.ContentType = contentType.MediaType;
        Response.StatusCode = StatusCodes.Status200OK;
        await Response.Body.WriteAsync(encoded);
    }
}
```



Key take aways

Lessons learned

Building offline capable Progressive Web Apps

- Don't just rely on `window.navigator.online`, actual test requests
- Not all data is equally cacheable, choose your strategy accordingly
- Working with requests & responses
 - Online & offline request routing has its differences
 - CORS will prevent caching if not set up correctly
 - CSS is not CORS compliant, neither are third party services
 - Caching opaque responses can be dangerous
 - Oauth `offline_access` != access event when offline
 - Instead disable silent renew and use cached token while offline
 - Mind Authorization headers on queued requests, they may need a refresh
- Working with structured data
 - CQRS: Not all data is equal, neither should the stack to work with it
 - Conflict is inevitable while working offline, consider using CRDTs on the write side
 - Event sourcing + commutative events == Grow-only set CRDT
 - Conflicts can be auto-resolved by reordering events (rebase)

Q&A

www.goeleven.com