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


Vijfsprong

DEMO

Introduction

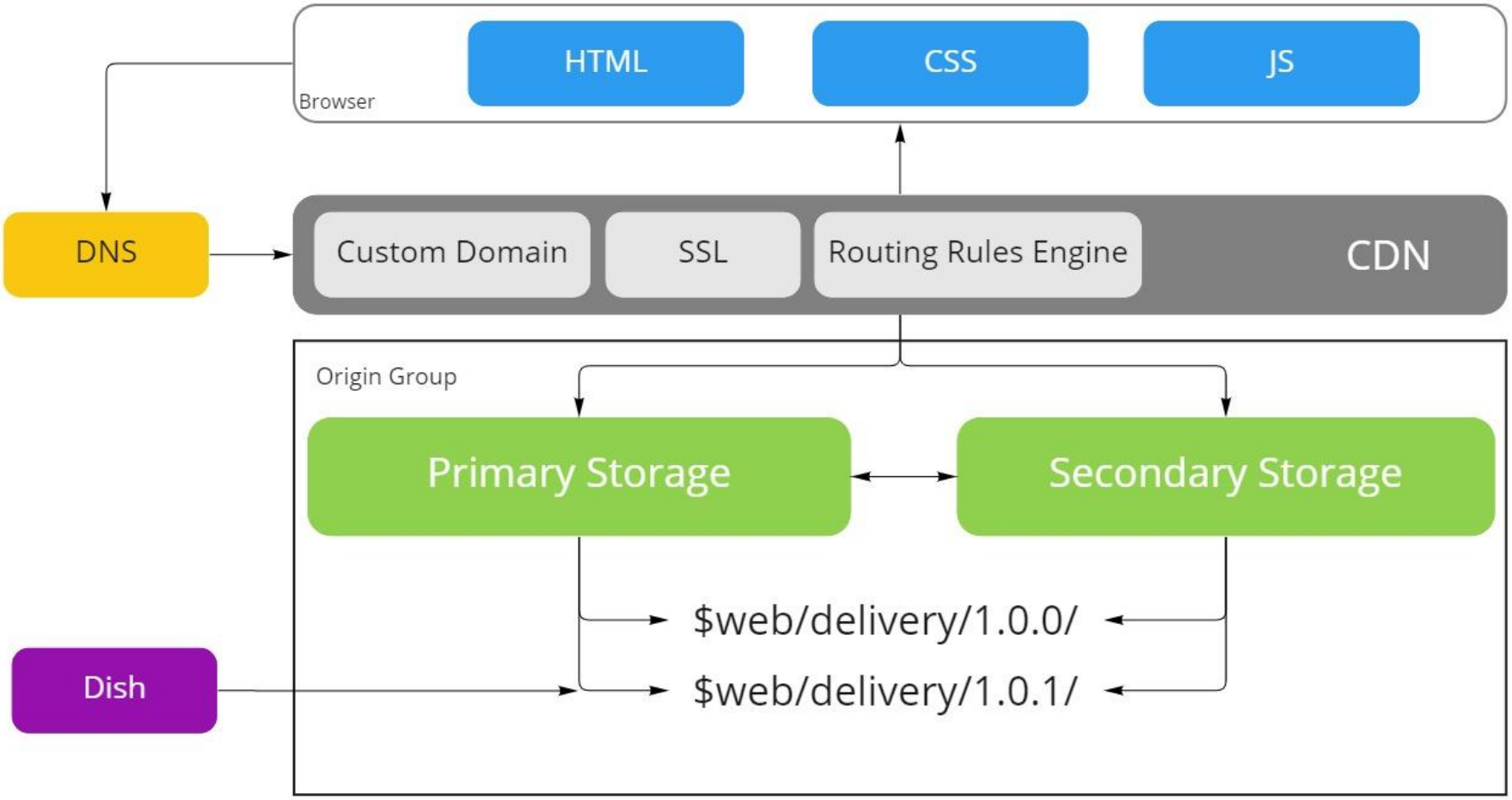
Architecture

Online or offline

Not all data is equal

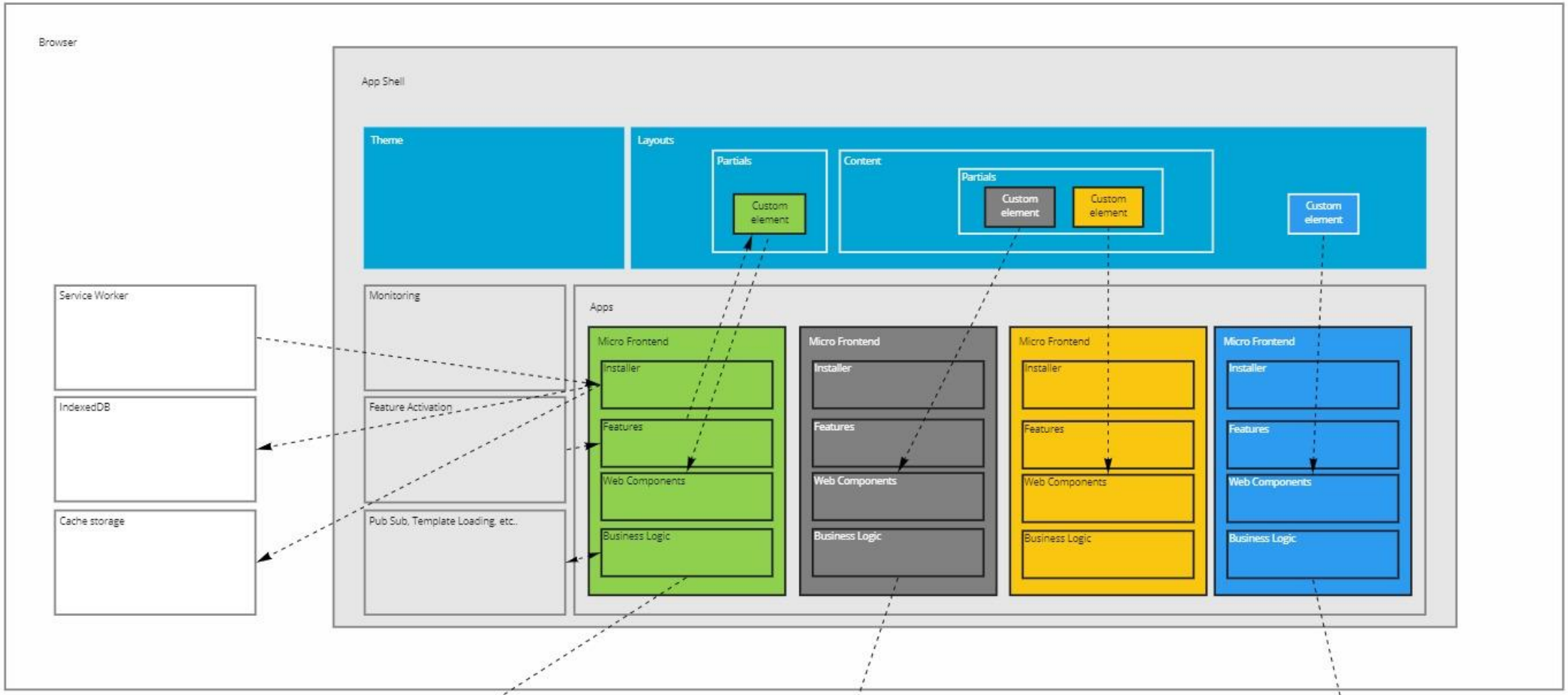
Hosted on CDN

Backed by geo redundant Azure Storage



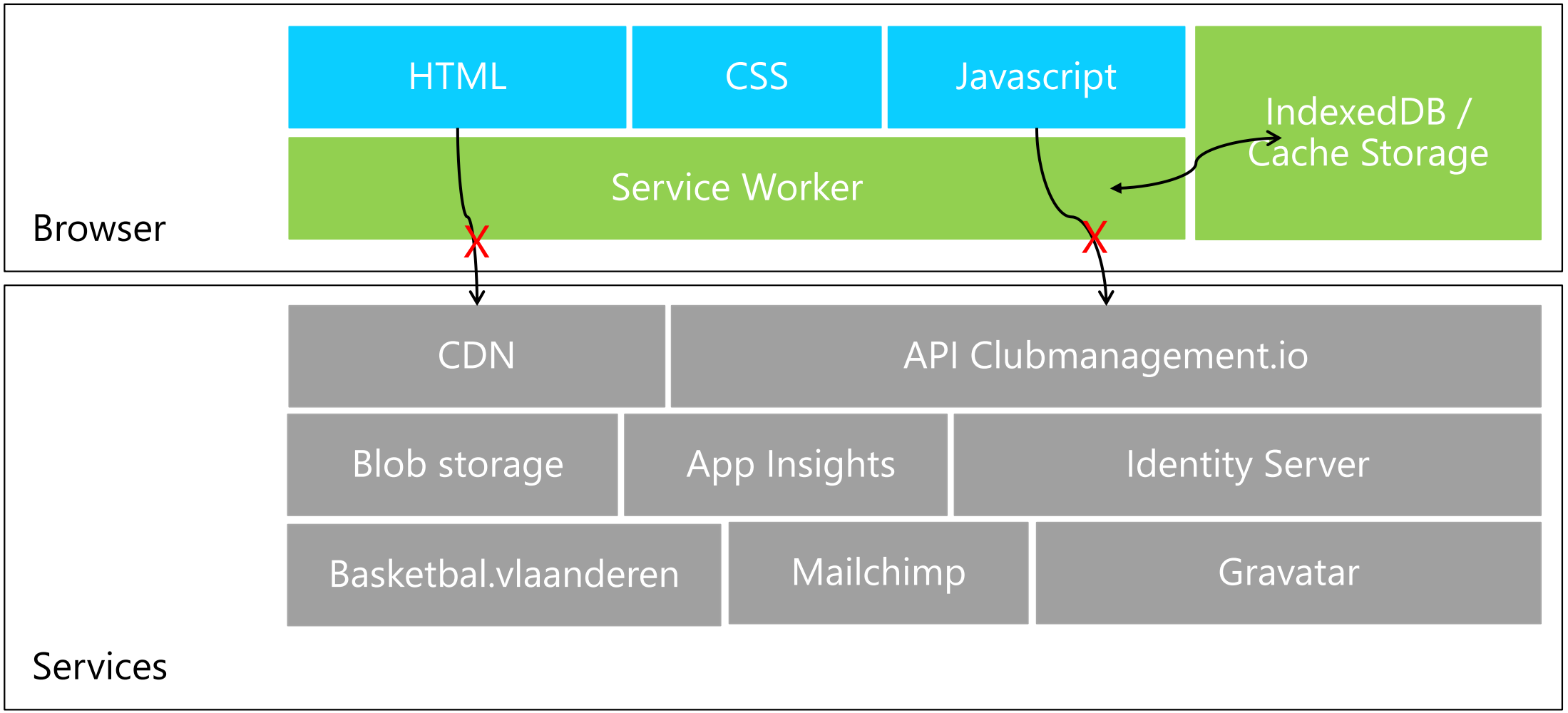
Front end architecture

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



How it works, even when offline

Service Worker intercepts and serves from IndexedDB or Cache storage

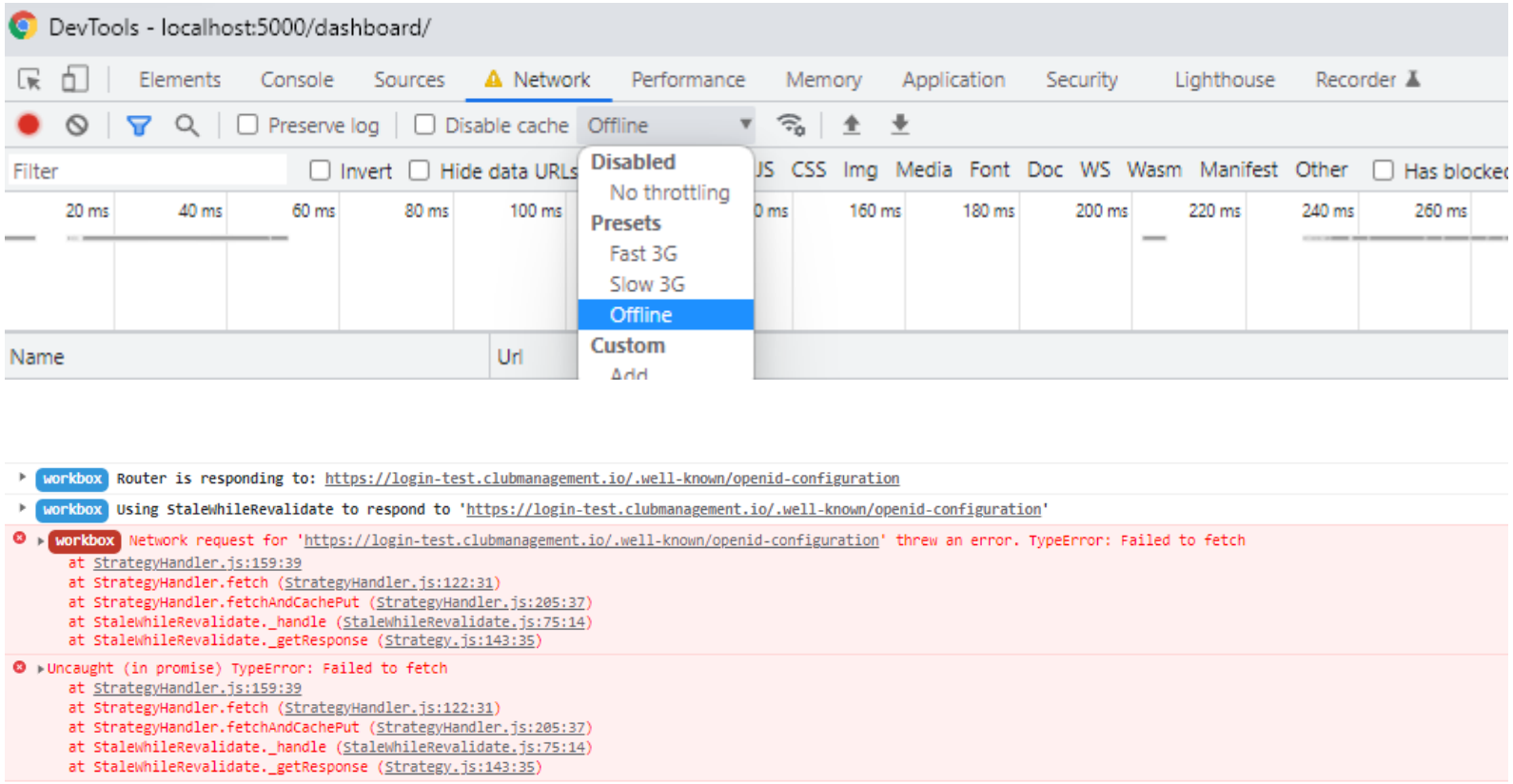


Introduction

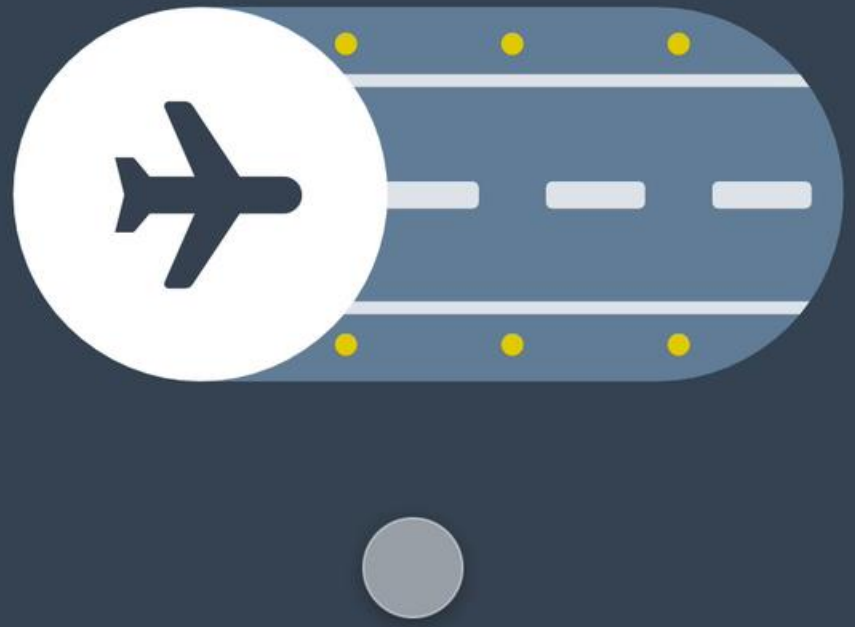
Architecture | **Online or offline** | Not all data is equal

Going offline

Dev tools, network inspector & console output



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
- Intent - Future - Uncertain outcome - Least stable	- Fact - Past - Certain outcome - Most stable	- 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

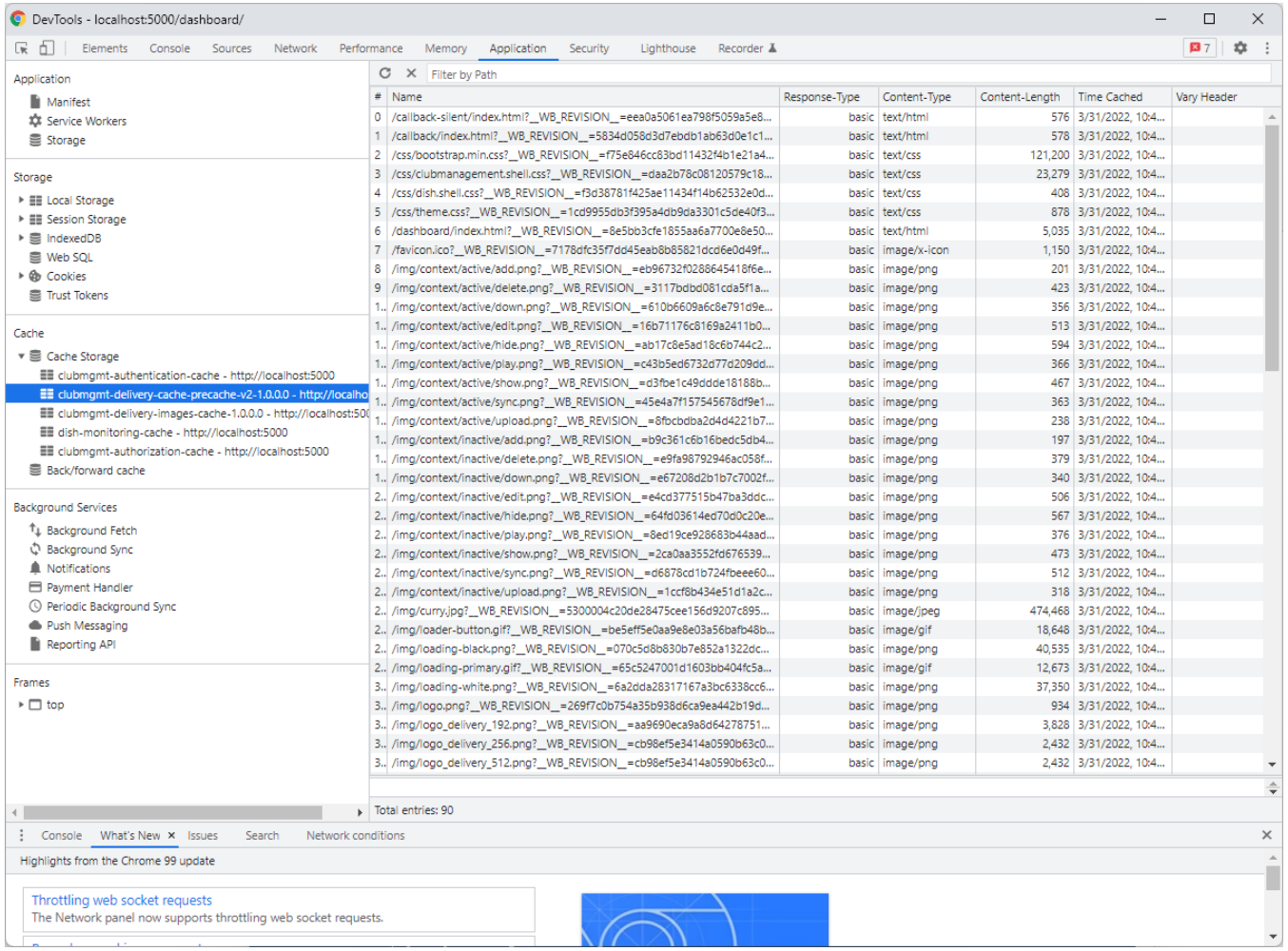
IndexedDB

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 || []);
```

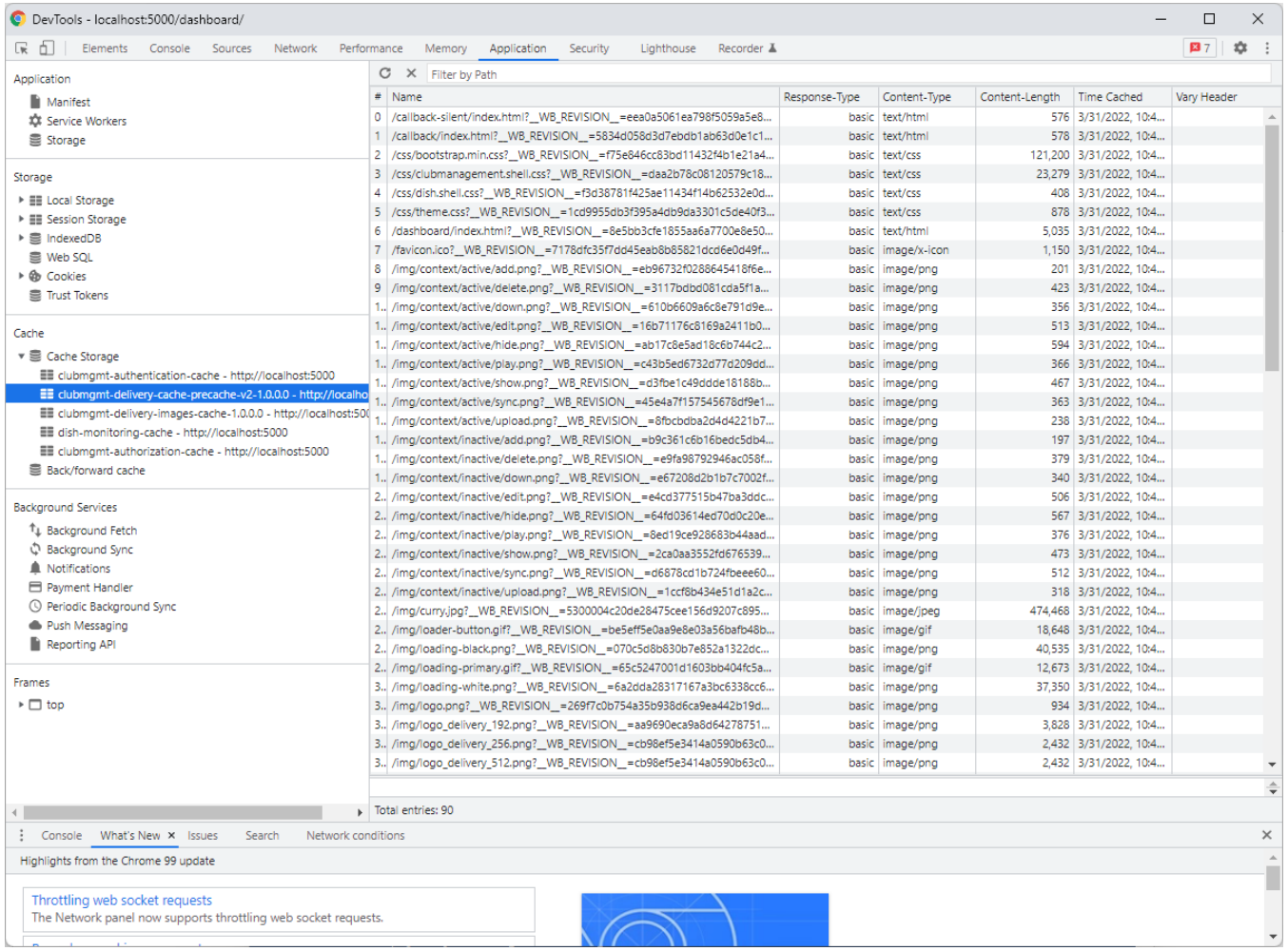
- Files must be versioned
  - Revision hashes (generated by Dish postprocessing step)
  - Service worker file must change for changes to take effect: build revision

```
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



Workbox will serve files ‘Cache first’

Request routing has its differences though…

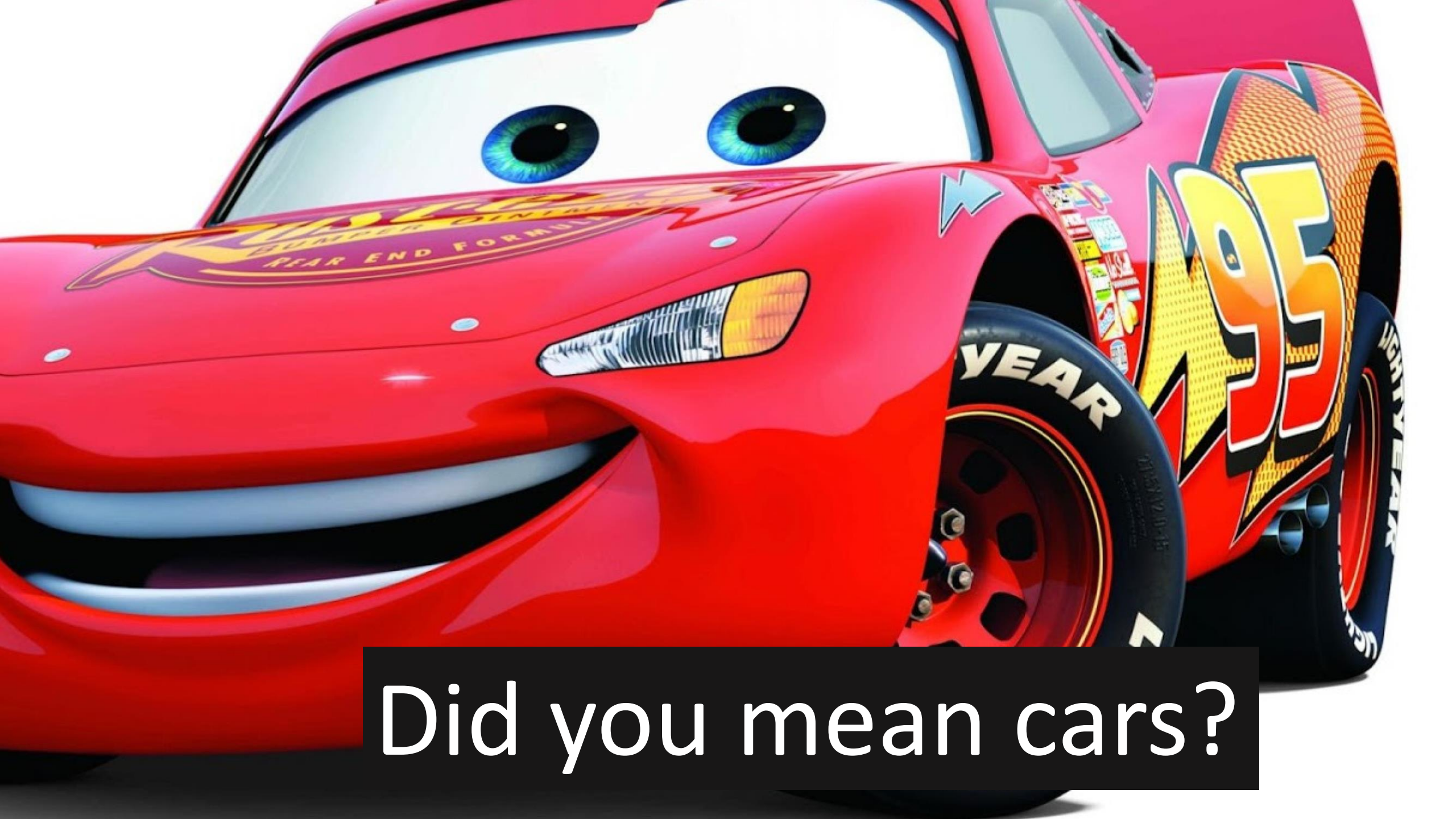
- File paths behave differently, '/dashboard' != '/dashboard/’
- Capitalization matters (case sensitive like linux, not insensitive as usually on windows)

- 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

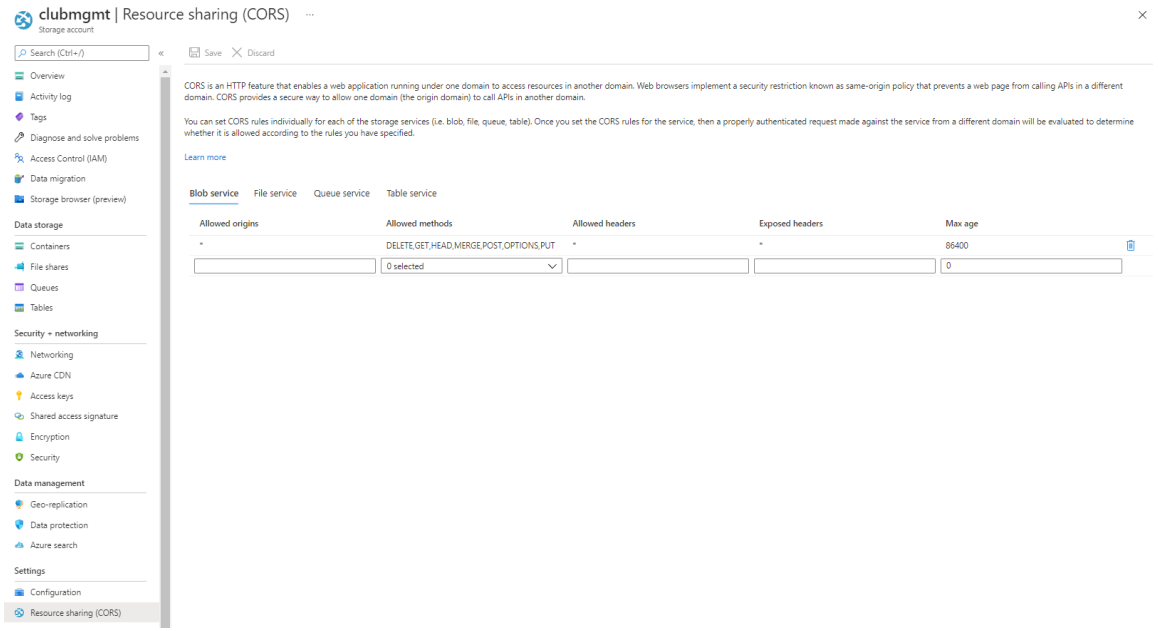
- Security feature which restricts scripts to interact with resources from other domains
- Opt-in on both client and server

Client

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

- Must match
  - Origin
  - Method
  - Header

Server (blob storage)



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]
              })
           ]
        })
     );
```

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

Working with requests & responses

Workbox

Files

CORS

OAuth

Queueing Requests

Oauth

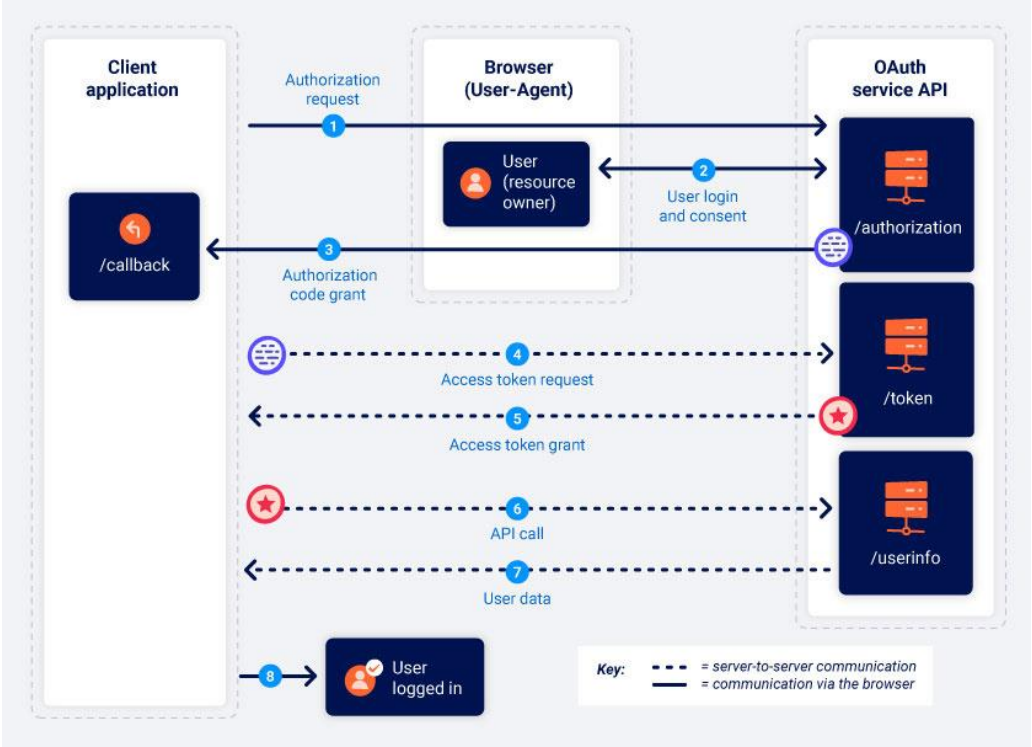
Open standard authorization protocol

- 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…

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

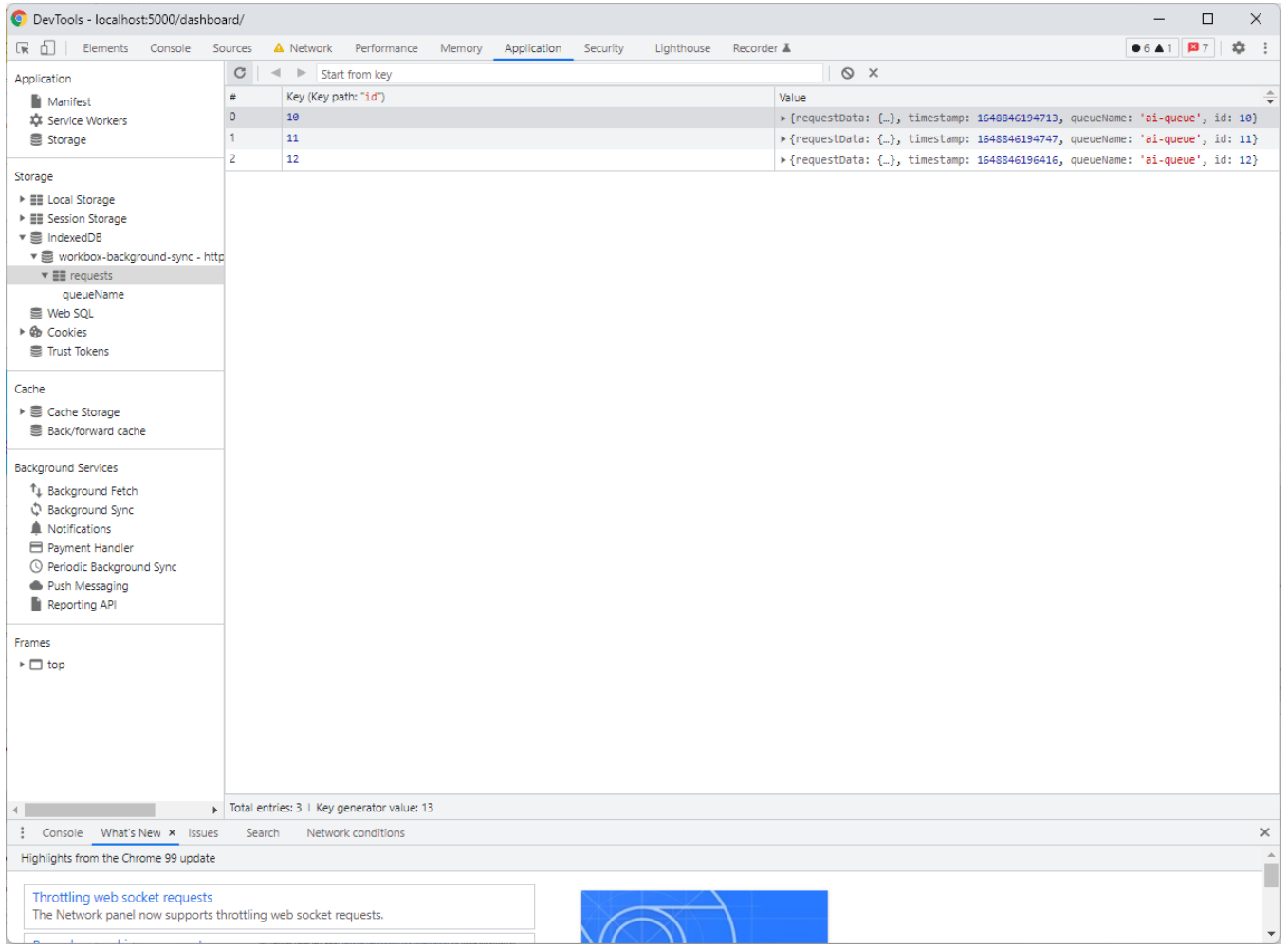
Add failed requests to a queue

- 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



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'
);
```

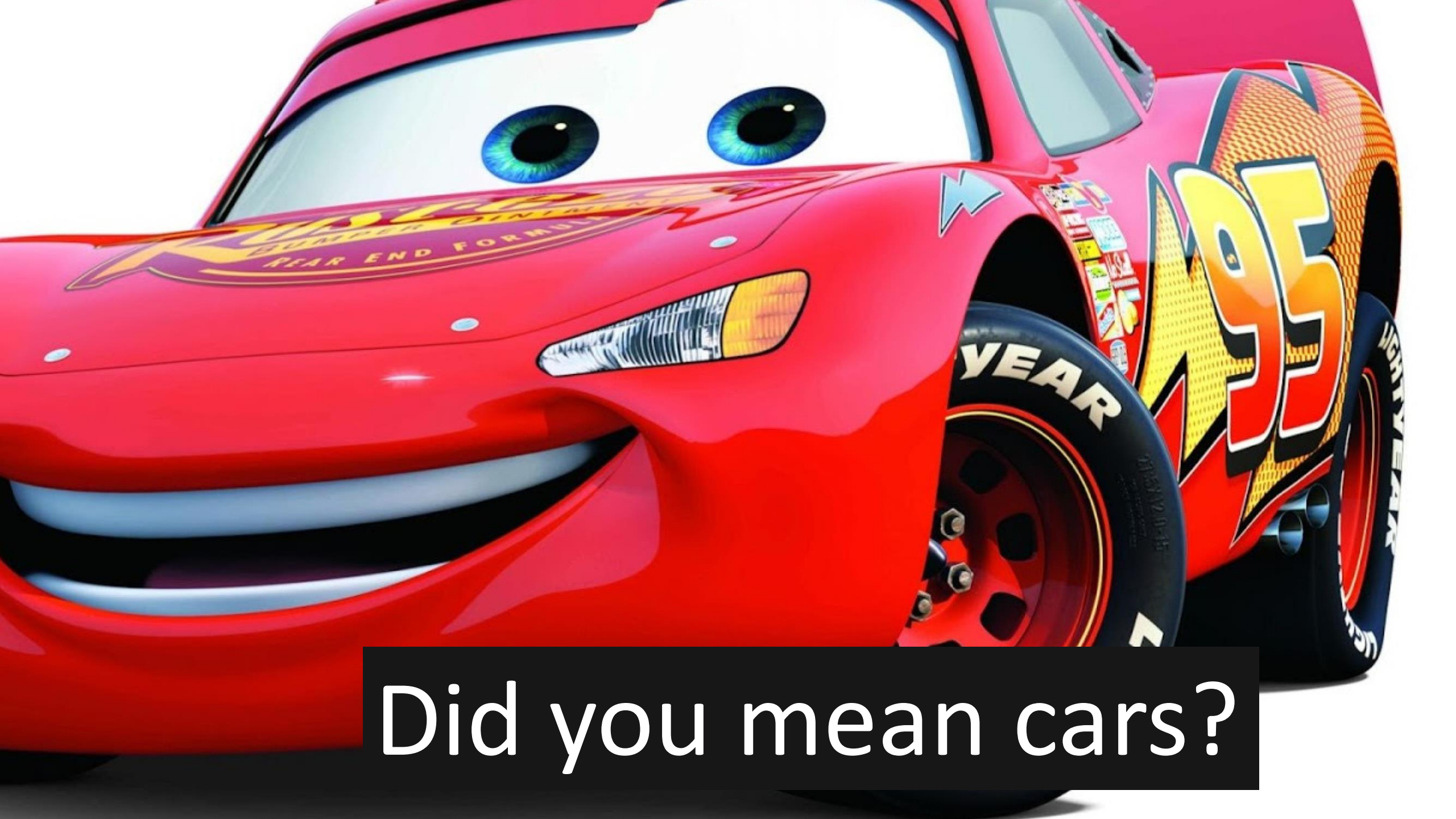
Replay when back online

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

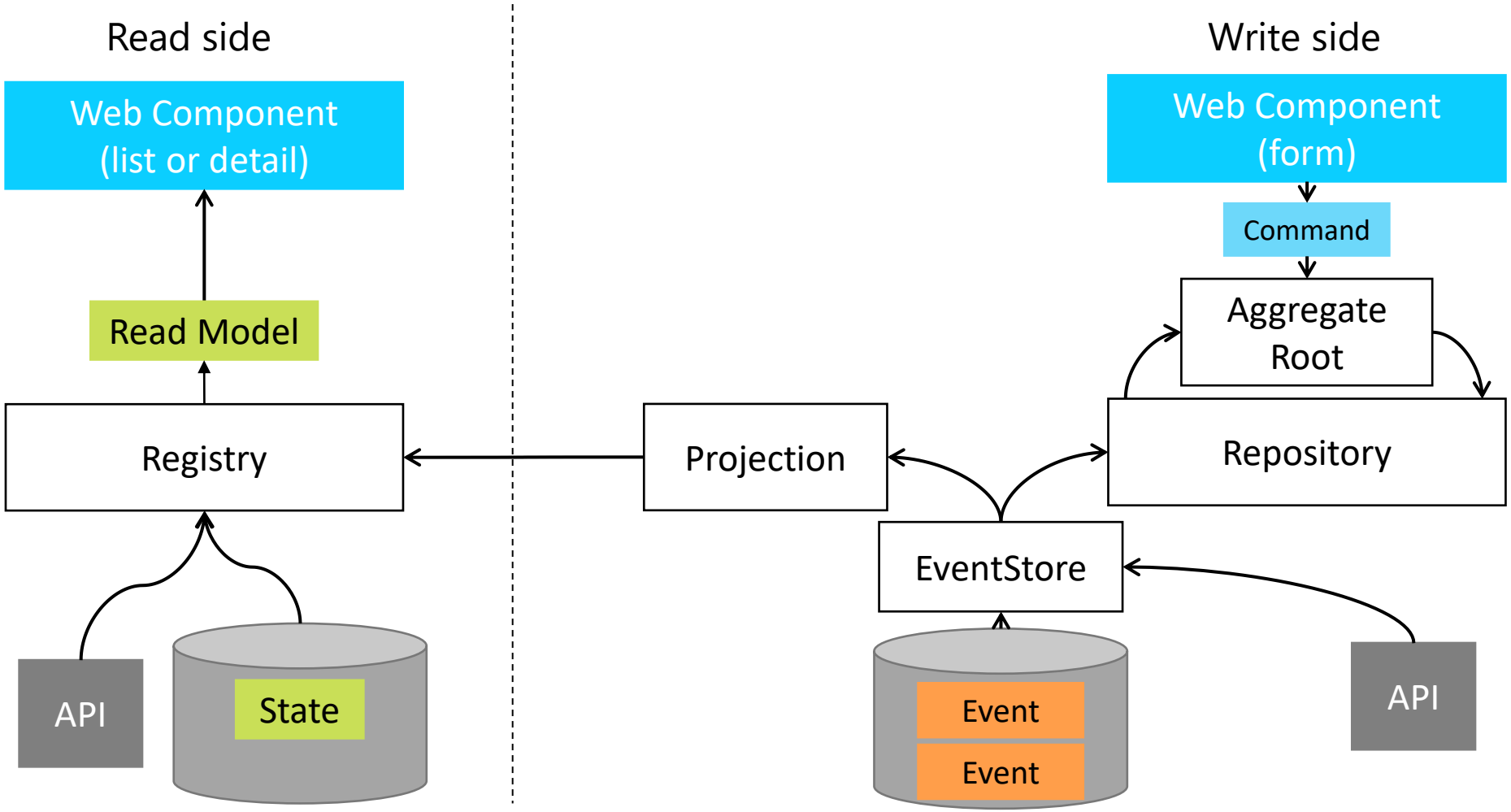
CQRS read	Inevitable Conflict	CRDTs
Conflict Handling	CQRS write	Synchronization



Did you mean cars?

Command Query Responsibility Segregation

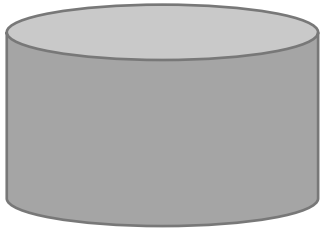
Different design for reads and writes



Read Side Example

Setting up indexeddb

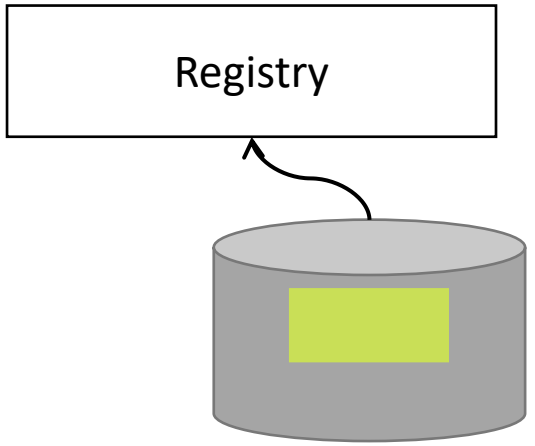
```
class IndexedDBFactory{
     open(name, version, onupgrade){
        return new Promise((resolve, reject) => {
            var request = indexedDB.open(name, version);
            request.onerror = function(event) { reject(); };
            request.onsuccess = function(event) {
                var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

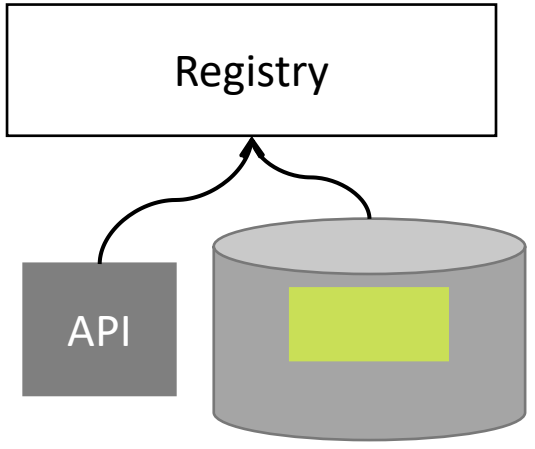
```
class Registry{

 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

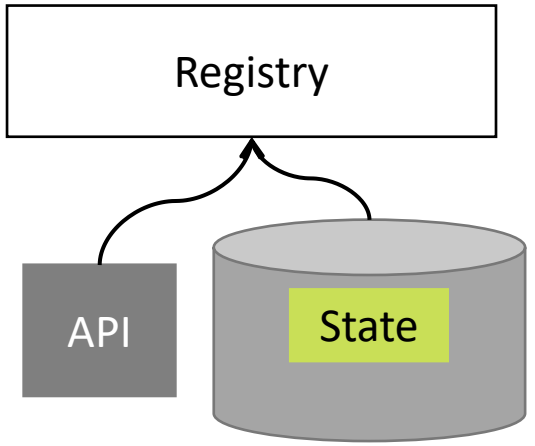
```
class Registry{

    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

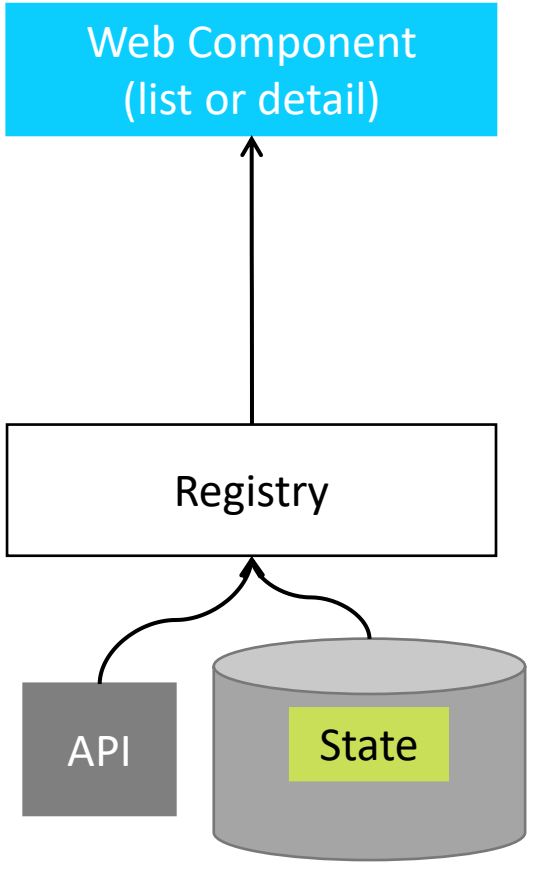
```
class Registry{

    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

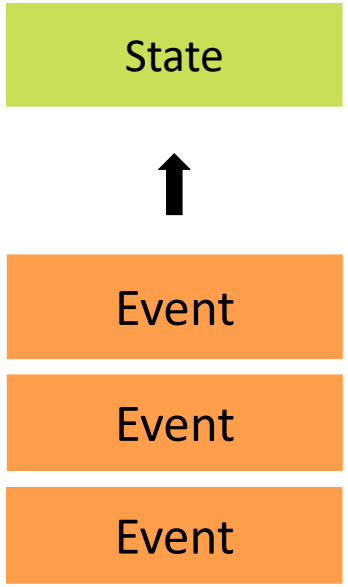
- Known CRDTs
  - Grow-only counter
  - Positive-Negative counter
  - **Grow-only set**
  - Two-phase set
  - Last-Write-Wins-Element-Set
  - Observed-Remove-Set
  - Sequence

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

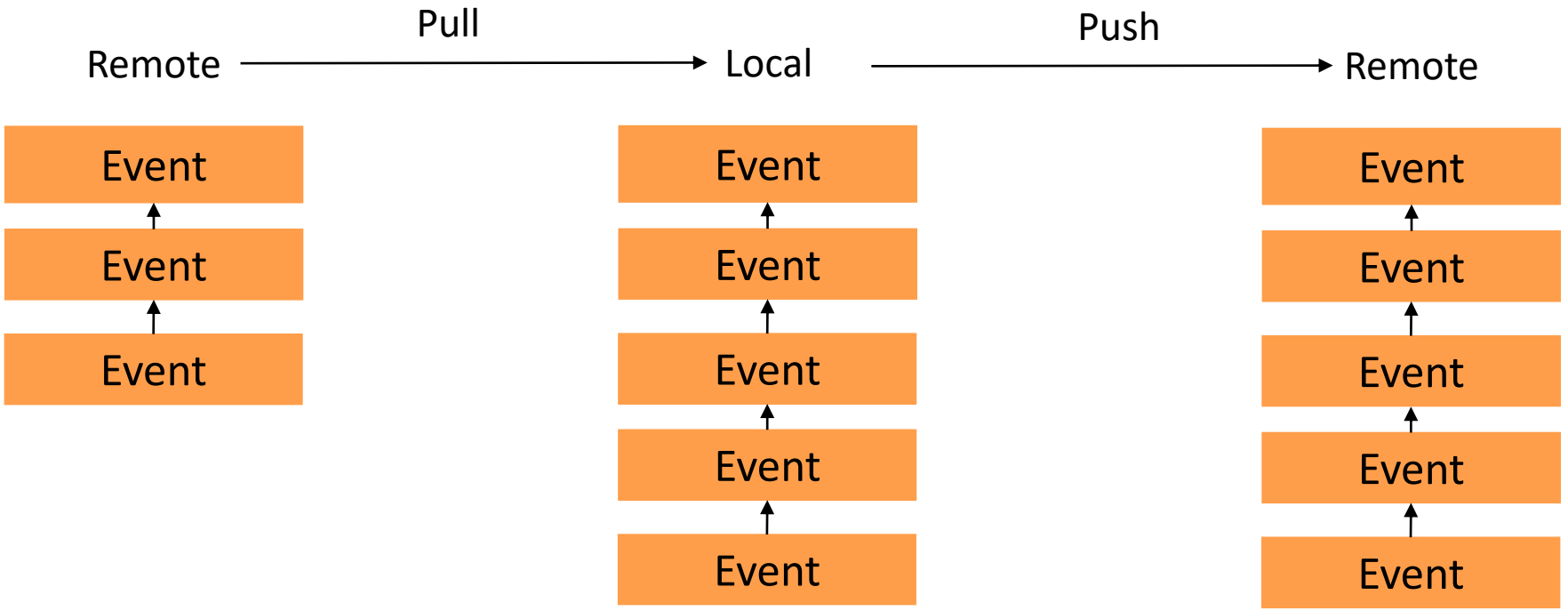
Conflict Handling

CQRS write

Synchronization

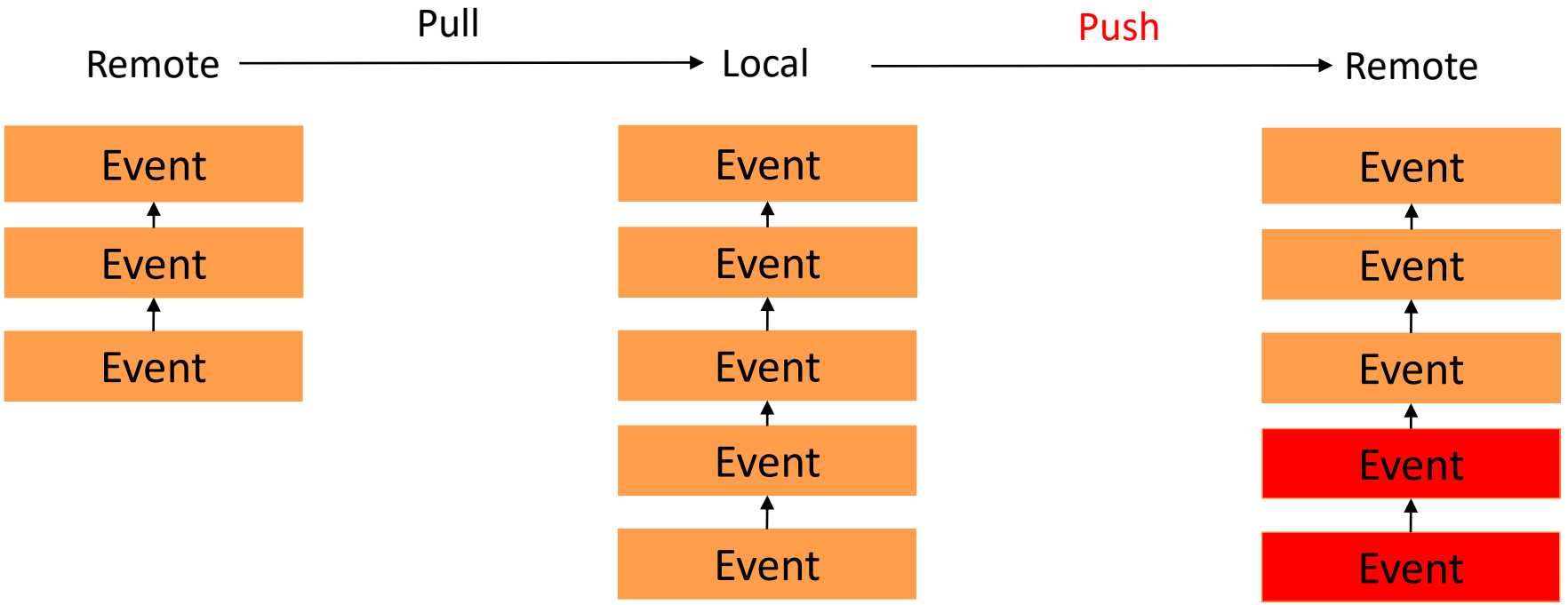
Conflict Resolution Inspired By Git

Pull & Push



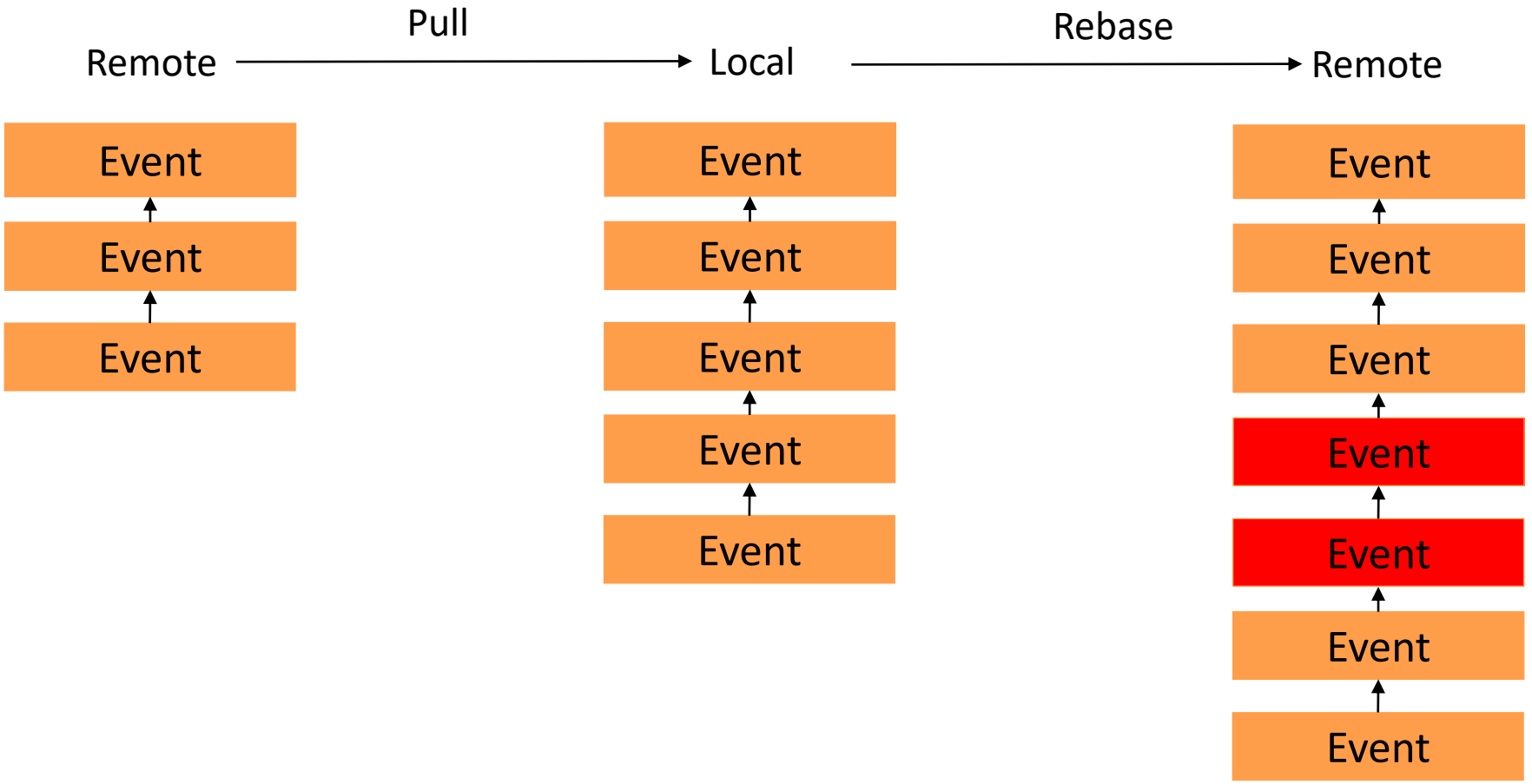
Conflict Resolution

Push will be rejected when conflicts



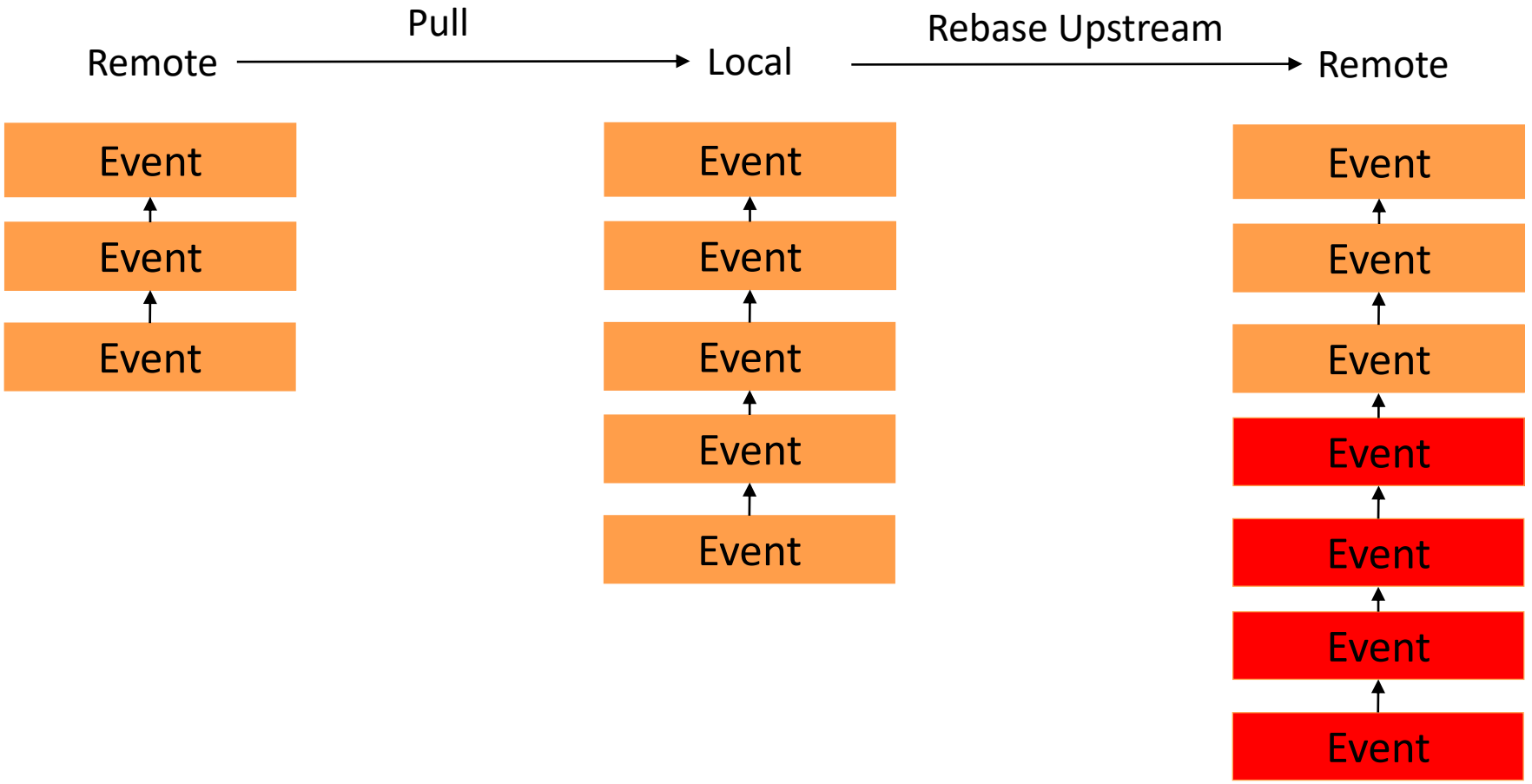
Conflict Resolution

Commutative = Order does not matter => Rebase



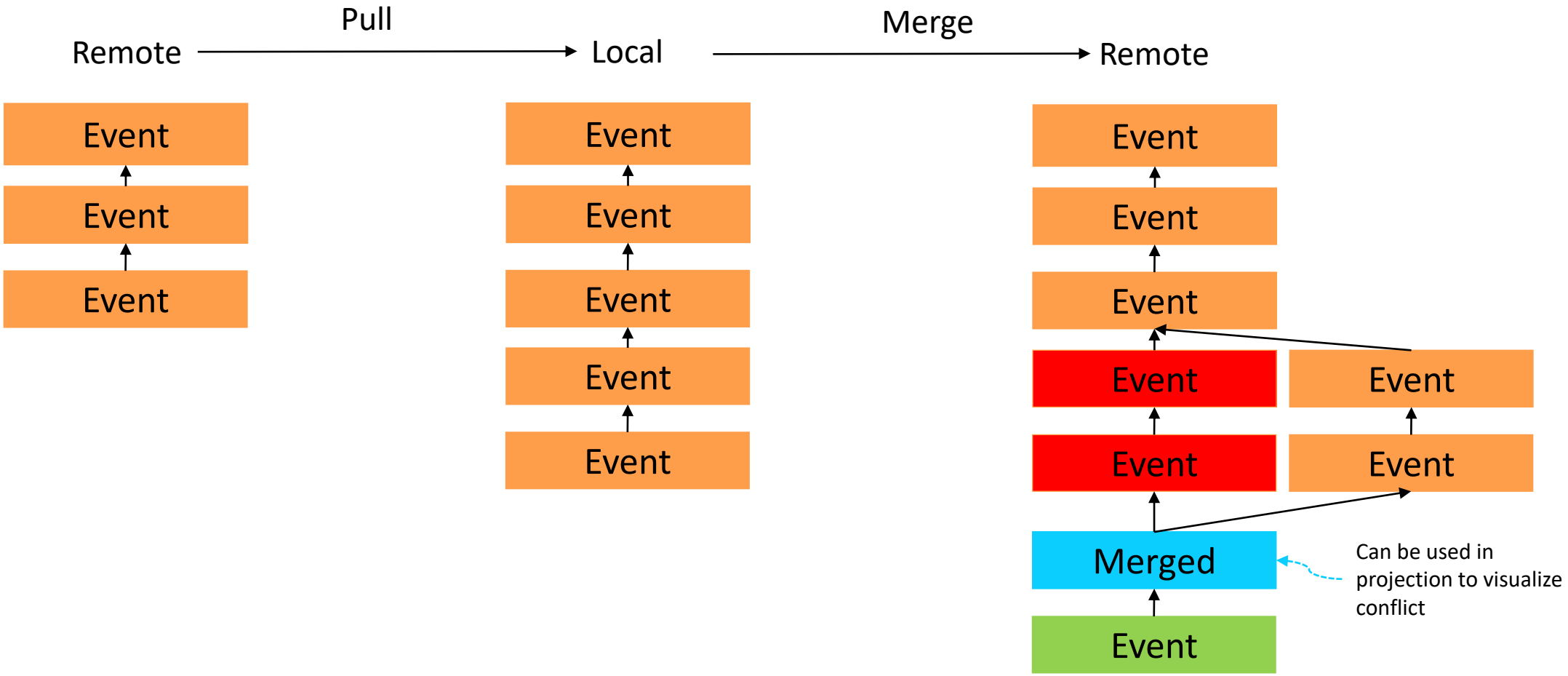
Conflict Resolution

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

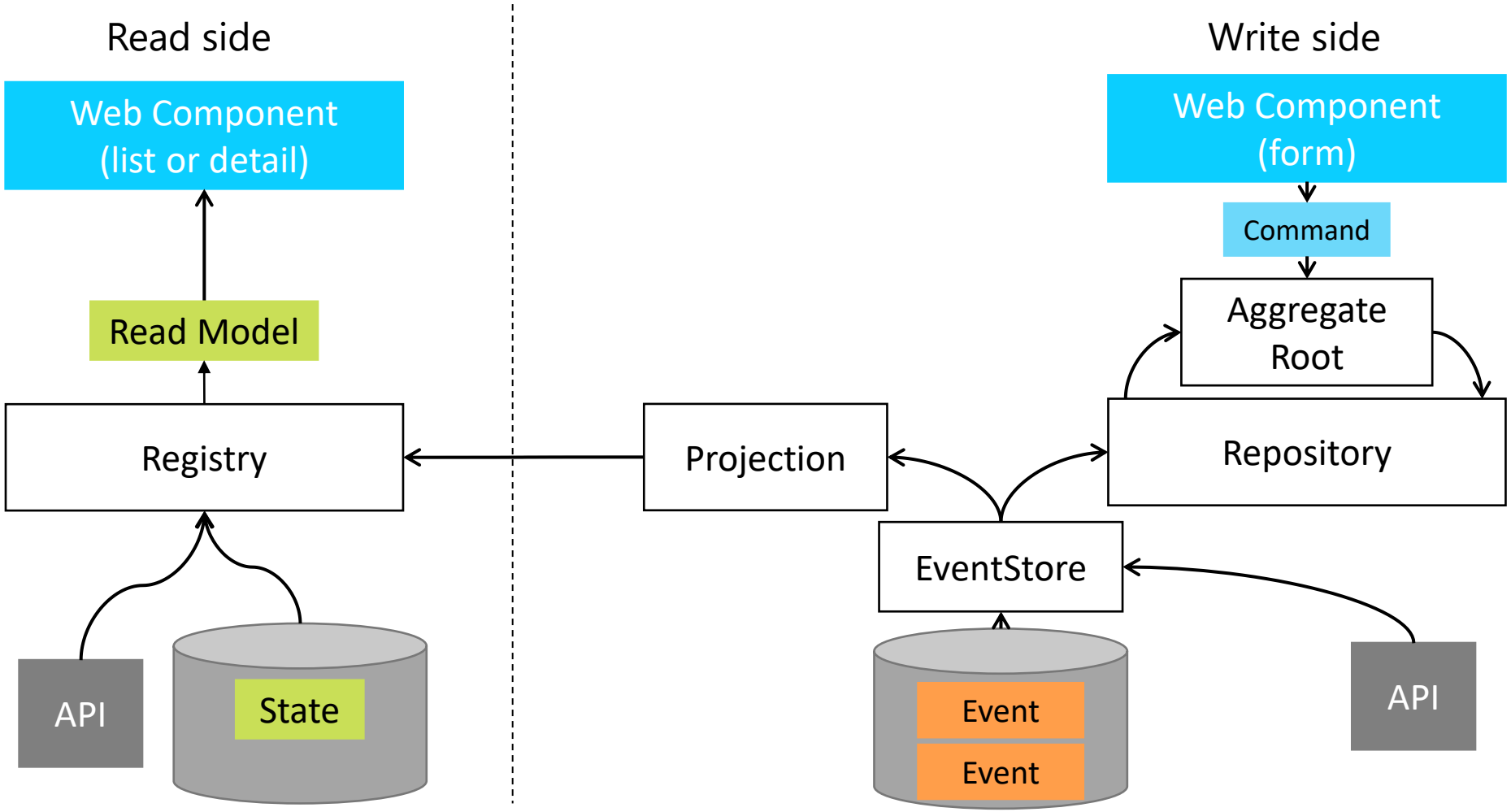
Conflict Handling

CQRS write

Synchronization

CQRS Write side

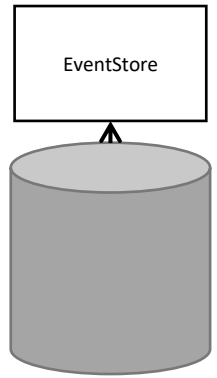
Working offline



Write Side Example

Event Store tables

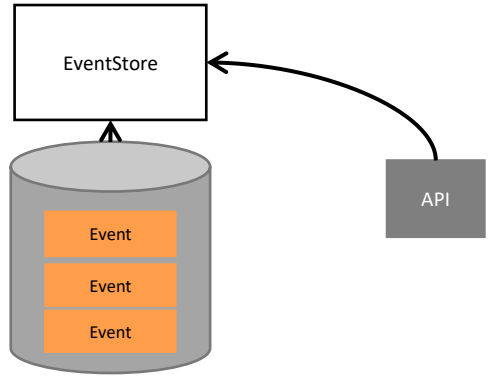
```
class EventStore{

    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

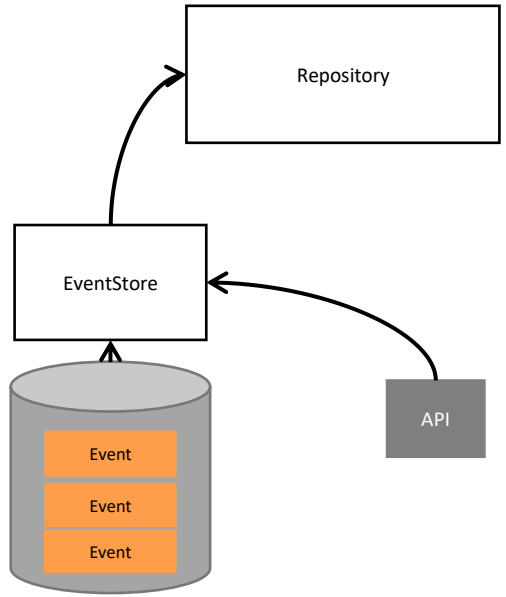
```
class OrderBookingsEventStore{

    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

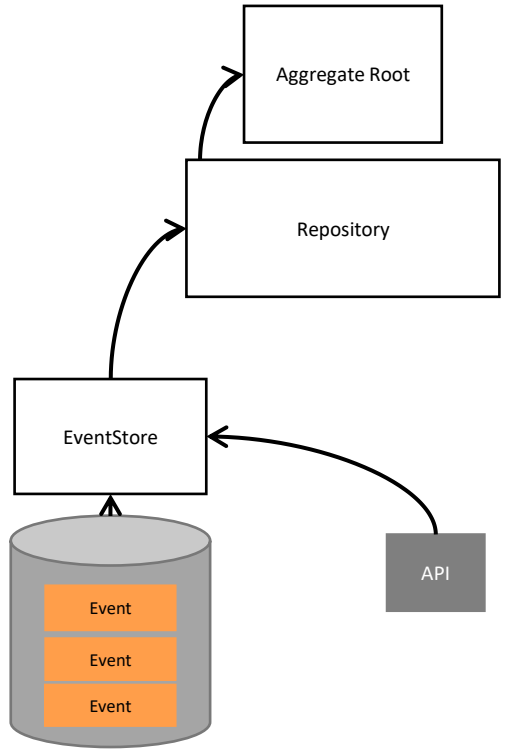
```
class OrderBookingRepository{

    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

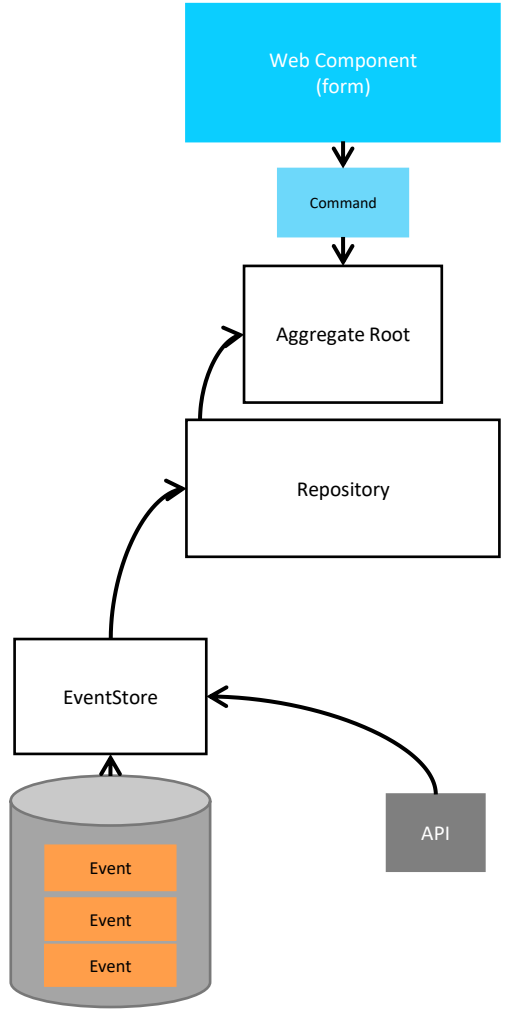
```
class OrderBooking extends EventSourced{

    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

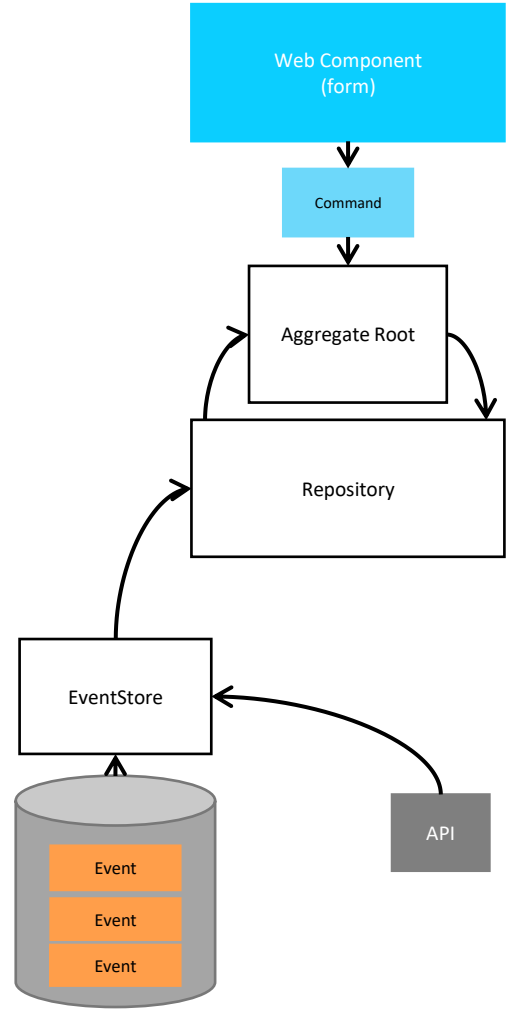
```
class BookOrder extends HTMLElement {

  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

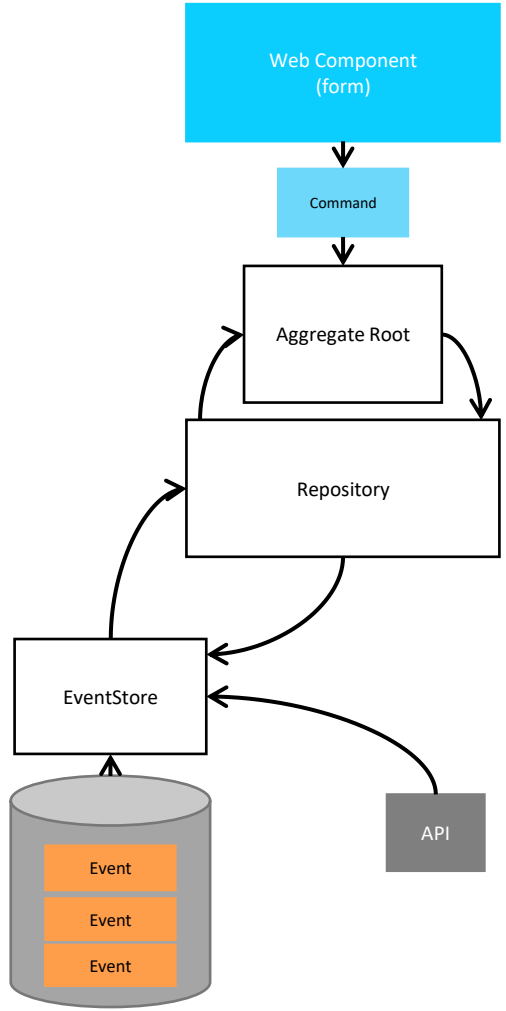
```
class OrderBooking extends EventSourced{

    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

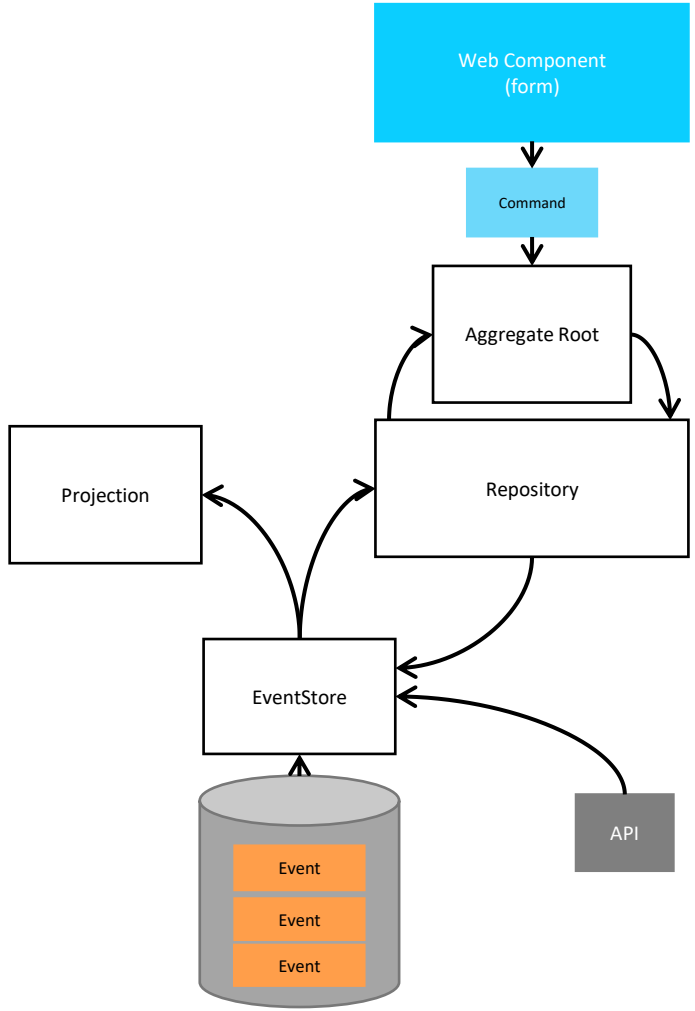
```
class OrderBookingRepository{

    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

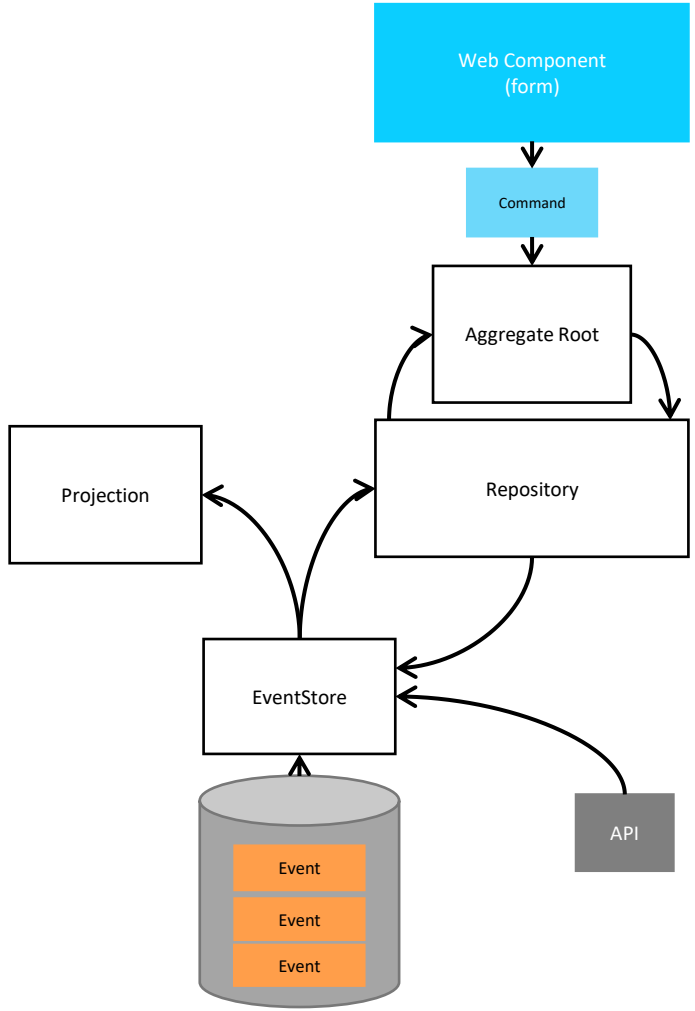
```
class OrderBookingsProjection{

    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

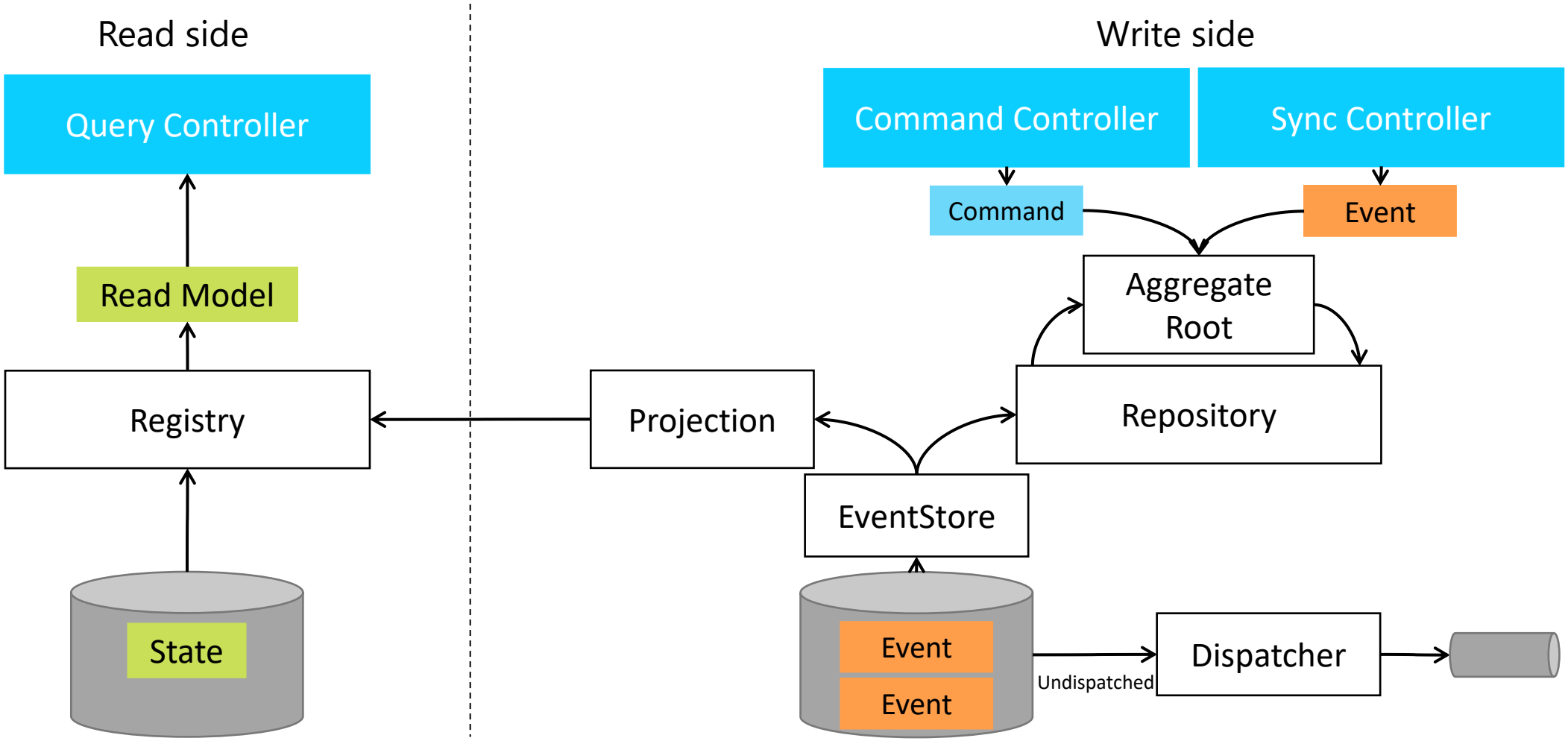
Conflict Handling

CQRS write

Synchronization

API Architecture

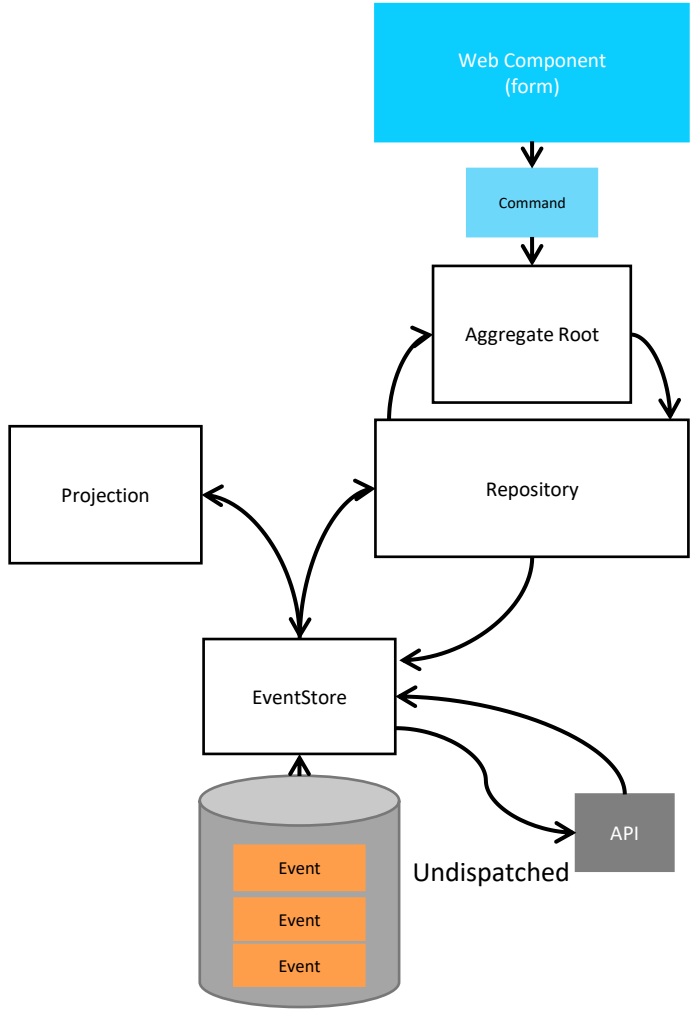
Same as the client-side architecture + sync, but written in C#



Synchronization

Frontend side

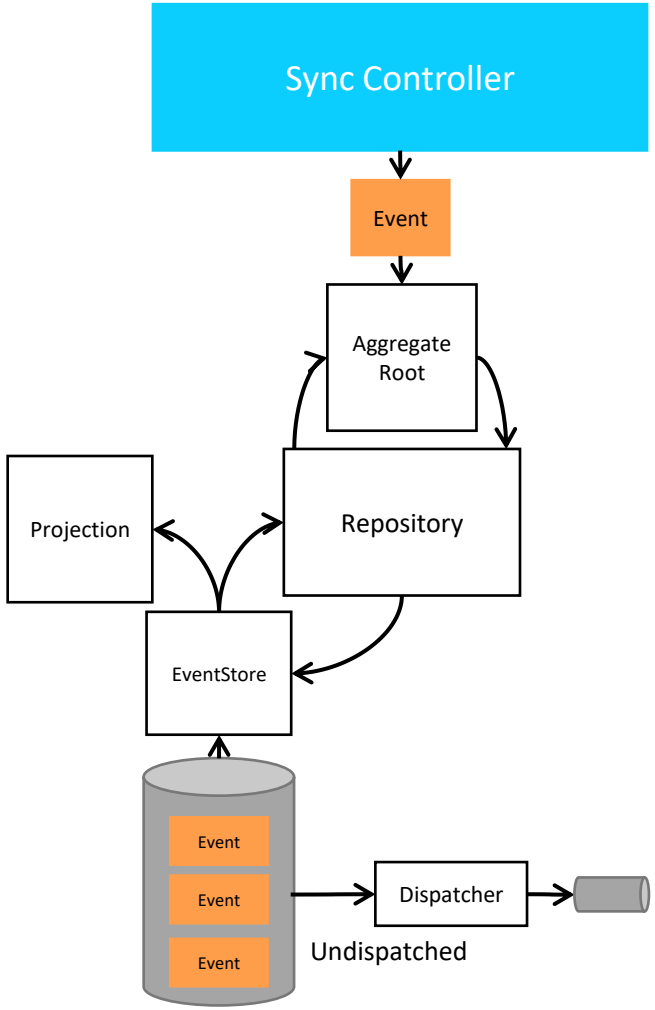
```
class OrderBookingsEventStore{
    async flush(tenantId){
        var entities = await this.eventStore.undispatched(tenantId);
        var entityIds = Object.keys(entities);
        if(entityIds && entityIds.length > 0){
            for (const id of entityIds) {
                var undispatched = 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