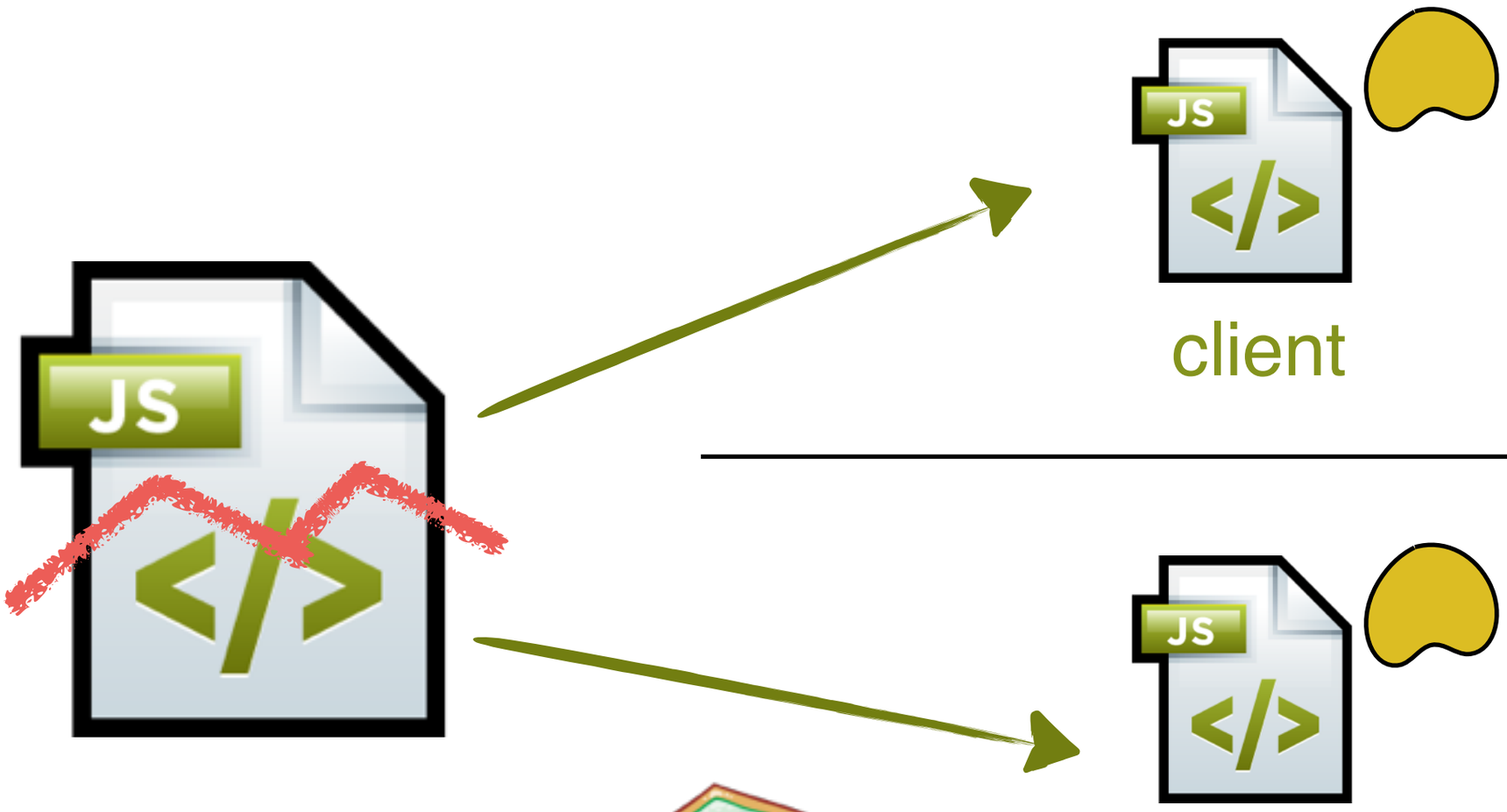
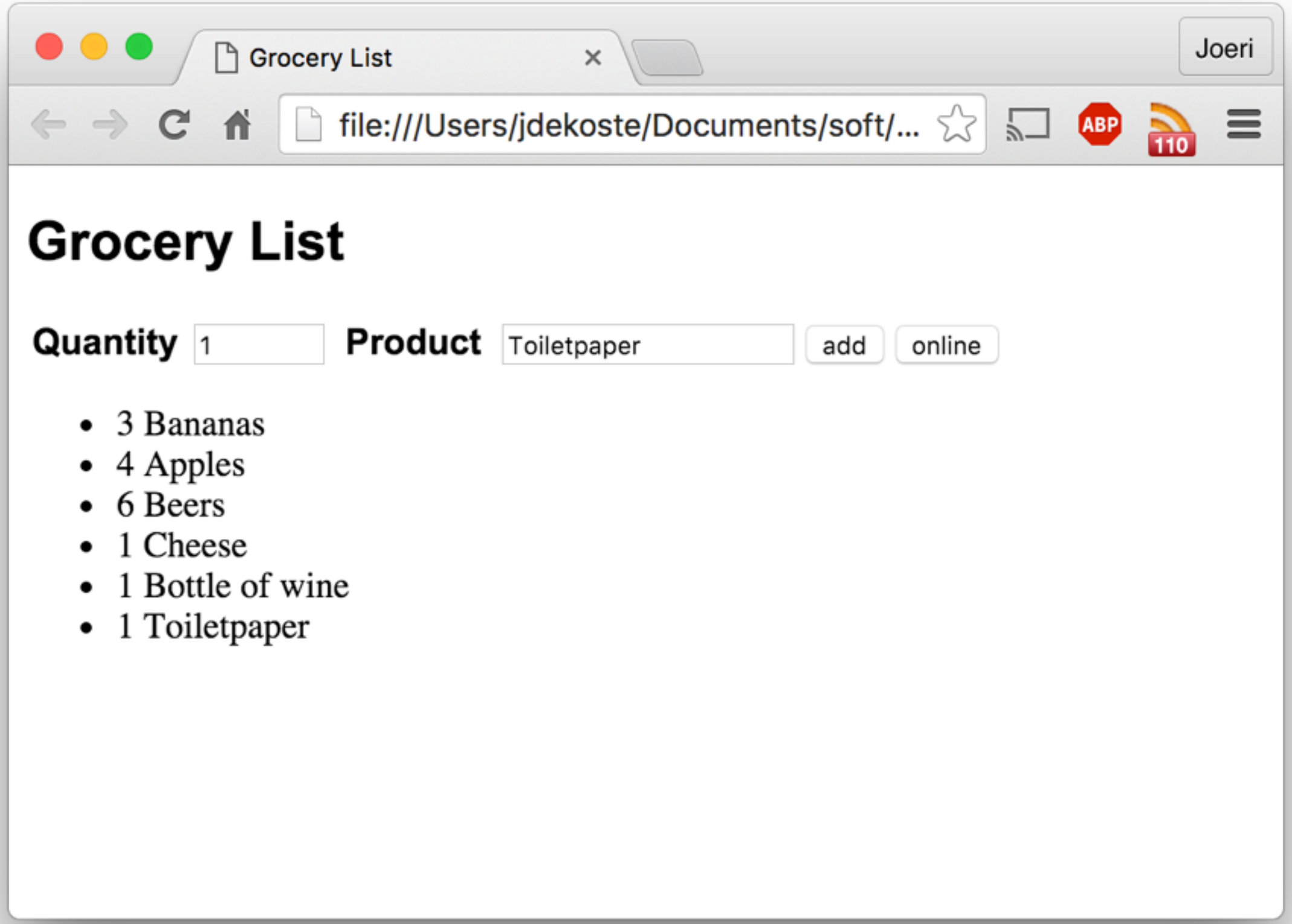
Implementing Offline Functionality for RIAs

Joeri De Koster
jdekoste@vub.ac.be

Offline functionality in Fearless



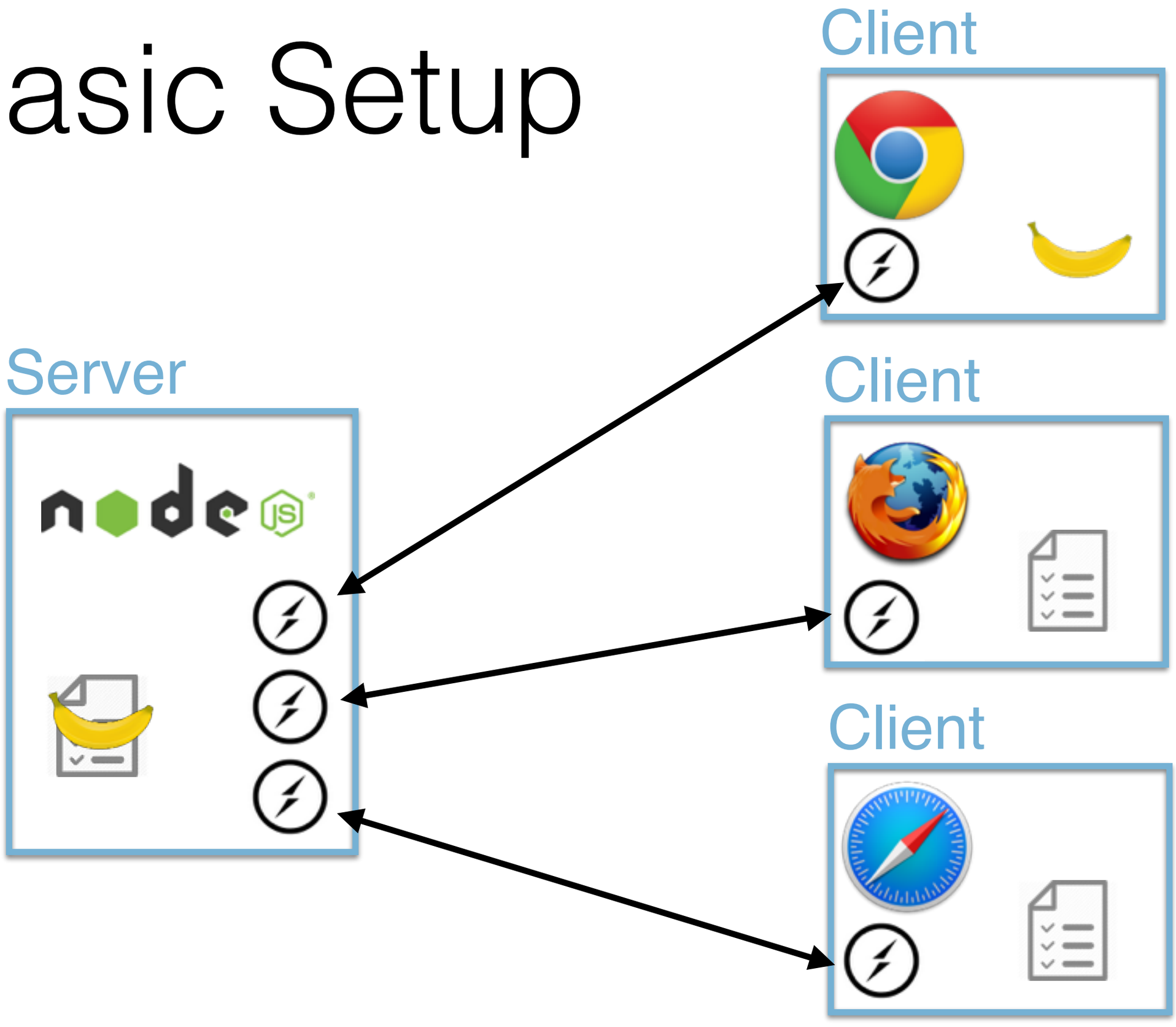
- Libraries
- Tool Support?
- Automatic slicing


Grocery List
Joeri
file:///Users/jdekoste/Documents/soft/...
ABP
110
Grocery List
Quantity 1
Product Toiletpaper
add
online
3 Bananas
4 Apples
6 Beers
1 Cheese
1 Bottle of wine
1 Toiletpaper

Requirements

- **Offline availability**
- **Persistence** of data between different sessions
- **Transparent access** for on/offline case
- **Consistency** between different clients

Basic Setup



server.js

```
1  var groceryList = {};
2
3  io.on("connection", function (socket) {
4
5    // Send current grocery list when client first connects
6    socket.on("connect_to_server", function (data) {
7      for (product in groceryList) {
8        socket.emit("add_product_to_client",
9                    { p : product, q: groceryList[product]});
10     }
11   });
12
13   // Broadcast added products to all clients
14   socket.on("add_product_to_server", function (data) {
15     var product = data["p"];
16     var quantity = data["q"];
17     groceryList[product] += Number(quantity);
18     io.emit("add_product_to_client", data);
19   });
20 });
```

server.js

```
1  var groceryList = {};
2
3  io.on("connection", function (socket) {
4
5    // Send current grocery list when client first connects
6    socket.on("connect_to_server", function (data) {
7      for (product in groceryList) {
8        socket.emit("add_product_to_client",
9                    { p : product, q: groceryList[product]});
10     }
11   });
12
13   // Broadcast added products to all clients
14   socket.on("add_product_to_server", function (data) {
15     var product = data["p"];
16     var quantity = data["q"];
17     groceryList[product] += Number(quantity);
18     io.emit("add_product_to_client", data);
19   });
20 });
```

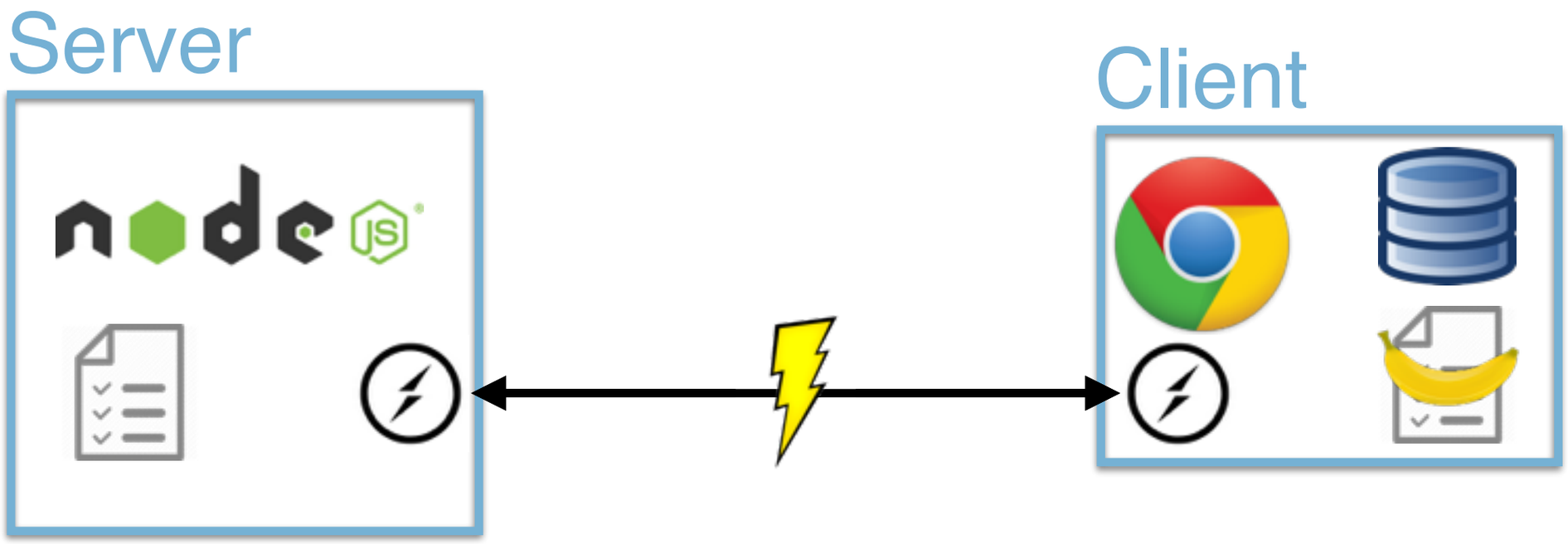
client.js

```
1  var groceryList = {};
2
3  // Initialising client and setting up socket
4  function initialise() {
5    socket = io();
6    socket.emit("connect_to_server");
7    socket.on("add_product_to_client", function (data) {
8      product = data["p"];
9      quantity = data["q"];
10     groceryList[product] += Number(quantity);
11     refreshDOM();
12   })
13 }
14
15 // Client adds product to grocery list
16 function addProduct() {
17   var product = document.getElementById("product").value;
18   var quantity = document.getElementById("quantity").value;
19   socket.emit("add_product_to_server", { p : product, q: quantity});
20 }
```

client.js

```
1  var groceryList = {};
2
3  // Initialising client and setting up socket
4  function initialise() {
5    socket = io();
6    socket.emit("connect_to_server");
7    socket.on("add_product_to_client", function (data) {
8      product = data["p"];
9      quantity = data["q"];
10     groceryList[product] += Number(quantity);
11     refreshDOM();
12   })
13 }
14
15 // Client adds product to grocery list
16 function addProduct() {
17   var product = document.getElementById("product").value;
18   var quantity = document.getElementById("quantity").value;
19   socket.emit("add_product_to_server", { p : product, q: quantity});
20 }
```

Manual Caching



client.js

```
1  var groceryList = {}
2  var groceryListCache = {}
3
4  function initialise() {
5    socket = io();
6    socket.emit("connect_to_server");
7    socket.on("add_product_to_client", function (data) {
8      product = data["p"];
9      quantity = data["q"];
10     groceryList[product] += Number(quantity);
11     refreshDOM();
12   })
13   window.addEventListener("online", flushCache);
14 }
15
16 function addProduct() {
17   var product = document.getElementById("product").value;
18   var quantity = document.getElementById("quantity").value;
19   if (navigator.onLine) {
20     socket.emit("add_product_to_server", { p : product, q: quantity});
21   } else {
22     groceryListCache[product] += Number(quantity);
23     refreshDOM();
24   }
25 }
26
27 function flushCache() {
28   for (product in groceryListCache) {
29     socket.emit("add_product_to_server",
30                 { p : product, q: groceryListCache[product]});
31   }
32   groceryListCache = {};
33 }
```

	Offline Availability	Persisted	Transparent Acces	Consistency
Manual Caching	✓	✗	✗	✗

Available Technologies

- Cookies
  - Limited in size (4KB)
  - Sent along with every http request
- Browser-specific solutions (e.g. Google Gears)
  - Not cross-platform
  - Outdated or deprecated
- Flash
  - Many users block Flash or require a click in order to enable flash content
  - Flash is notoriously unreliable

Persistent Storage: LocalStorage + AppCache

- Local Storage
  - At least 5MB of locally stored state per domain
  - All pages, from one origin, can store and access the same data
  - window.localStorage
  - window.sessionStorage
  - Data has to be manually converted to strings

- AppCache
  - Offline browsing, pages are available while offline
  - Speed, cached resources load faster
  - Reduced server load

client.js

```
1  var groceryListCache = {
2    add: function (product, quantity) {
3      var cache = JSON.parse(localStorage.groceryListCache);
4      cache[product] += Number(quantity);
5      localStorage.groceryListCache = JSON.stringify(cache);
6    },
7    empty: function () {
8      localStorage.groceryListCache = "{}";
9    }
10 };
```

client.appcache

```
CACHE MANIFEST
client.css
client.js
```

	Offline Availability	Persisted	Transparent Acces	Consistency
HTML	✓	✓	✗	✗

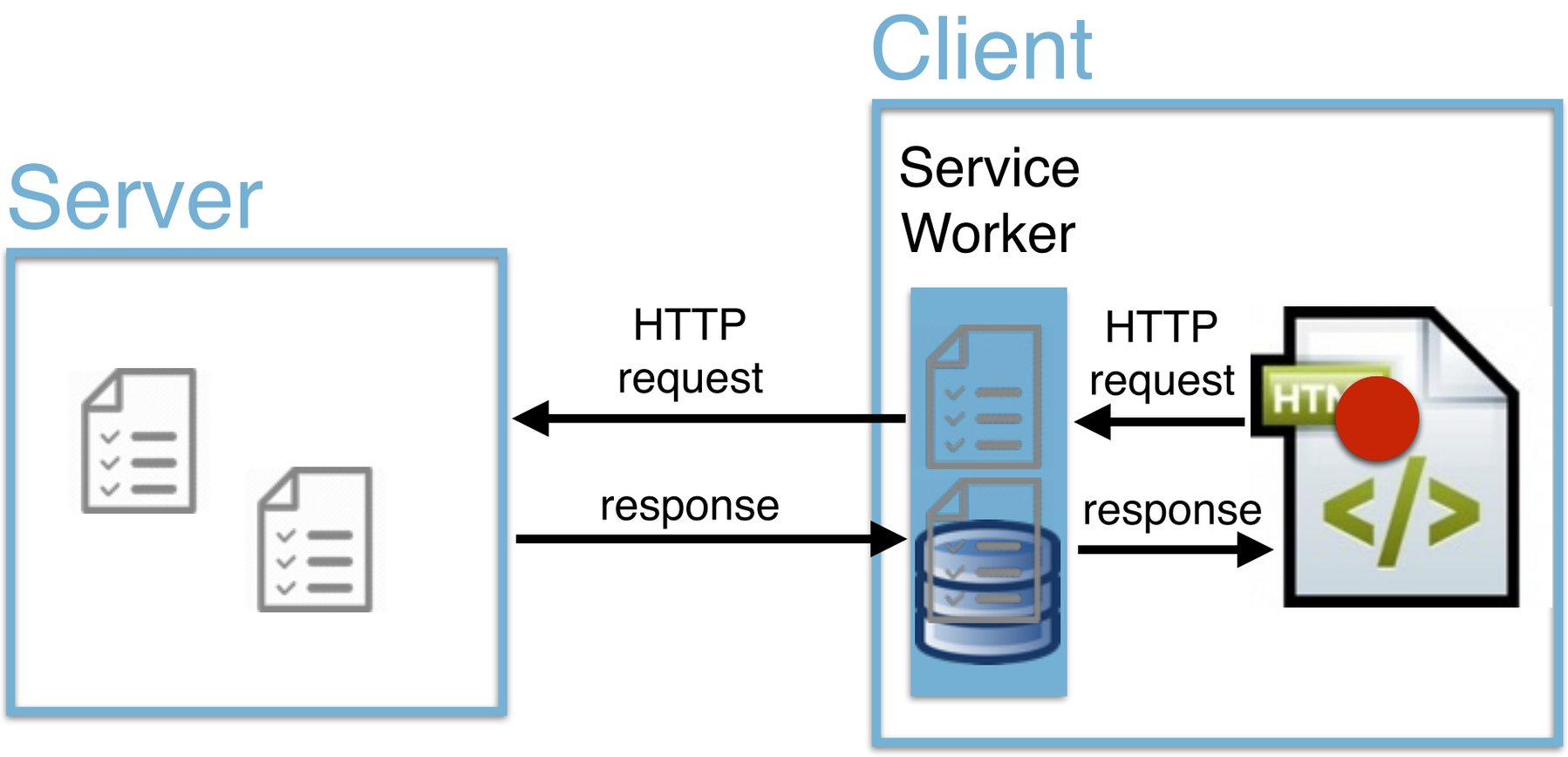
Local Storage

- ✗ Manual conversion of data to strings (JSON)

AppCache

- ✓ Offline availability, even when switching between pages
- ✗ No way to manage cached pages (changing the manifest file fetches all pages)
- ✗ Non-cached resources will not load on a cached page
- ✗ Google “Application Cache is a Douchebag”

Network Proxies: **ServiceWorkers**



client.js

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/service-worker.js');
}
```

service-worker.js

```
1  var CACHE_VERSION = "groceryListCache::v1";
2  var CACHE_NAME = "groceryListCache"
3
4  var groceryListCacheDelta = {}
5
6  this.addEventListener("install", event => {
7    event.waitUntil(
8      caches.open(cacheVersion).then(function (cache) {
9        return cache.addAll(["/client.html", "/client.js", "/client.css"]);
10     })
11   );
12 });
13
14 this.addEventListener("fetch", function (event) {
15   var request = event.request;
16   if (isAPICall(request.url)) {
17     var response = fetch(request).then(onlineResponse, offlineResponse);
18     event.respondWith(response);
19   } else {
20     var response = fetch(request);
21     event.respondWith(fetch(request));
22   }
23 });
24
25 function onlineResponse (response) {
26   caches.open(CACHE_VERSION).then(function (cache) {
27     cache.put(CACHE_NAME, response);
28   });
29   flushCacheDelta();
30   return response;
31 }
32
33 function offlineResponse () {
34   var product = getProduct(request.url);
35   var quantity = getQuantity(request.url);
36   groceryListCacheDelta[product] += Number(quantity);
37   return caches.open(CACHE_VERSION).then(function (cache) {
38     return cache.match(CACHE_NAME).then(function (cached) {
39       return cached.json().then(function (groceryList) {
40         var jsonString = JSON.stringify(merge(groceryList, groceryListCacheDelta));
41         var blob = new Blob([jsonString], { type: 'application/json' });
42         return new Response(blob);
43       });
44     });
45   });
46 }
```

- Extensive use of promises
- Event-driven code
- Manual conversion

	Offline Availability	Persisted	Transparent Acces	Consistency
Service Workers	✓	✓	✗ ✓	✗

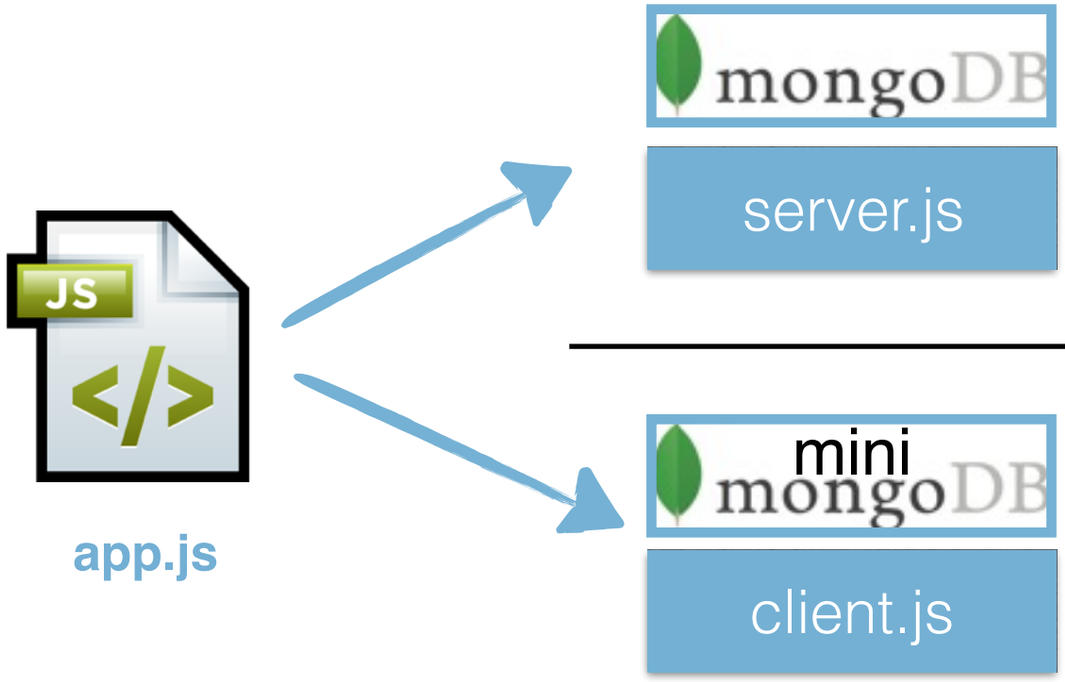
✓ Solves some of the issues with AppCache

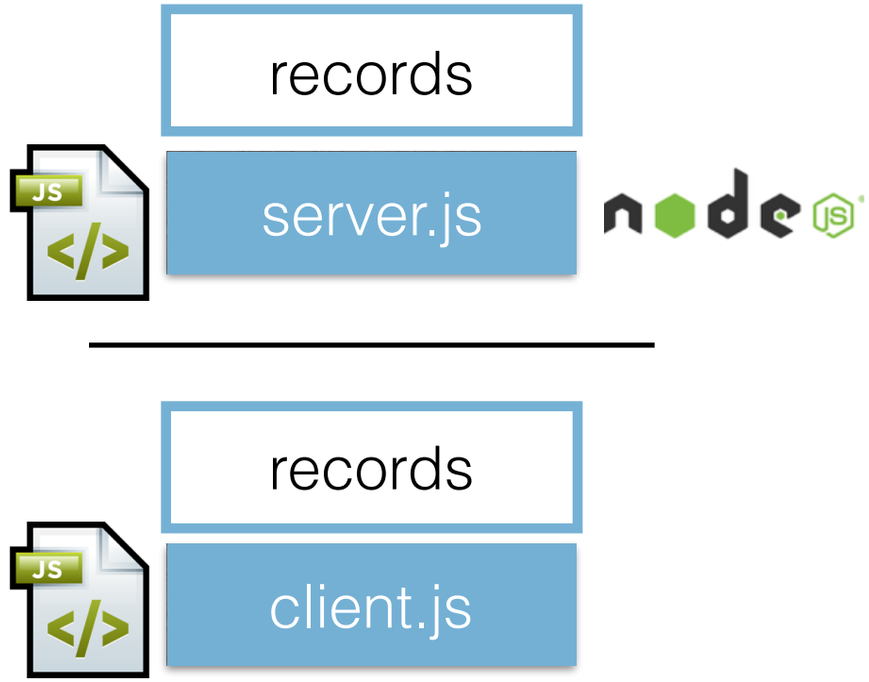
✗ Manipulating response is complex

Synchronised Collections

METEOR

deepstream.io





grocerylist.js

```
1  var GroceryList = new Mongo.Collection("grocerylist");
2
3  if (Meteor.isClient) {
4    // This code only runs on the client
5    Template.body.events({
6      "submit .addProduct": function (event) {
7
8        // Get value from form element
9        var p = event.target.product.value;
10       var q = event.target.quantity.value;
11
12       // Insert a grocery item into the collection
13       GroceryList.insert({
14         product: p,
15         quantity: q
16       });
17     }
18   });
19 }
```

grocerylist.html

```
<ul>
  {{#each grocerylist}}
    {{> item}}
  {{/each}}
</ul>

<template name="item">
  <li>{{quantity}} {{product}}</li>
</template>
```

	Offline Availability	Persisted	Transparent Acces	Consistency
METEOR	✓	✓	✓	✗

- ✓ Changes are automatically pushed to the server
- ✗ Meteor: big framework, not easy to add to existing code

Eventual Consistency

Swarm.js

- Conflict-free replicated data type (CRDT)
- Concurrent changes merge automatically
- Offline replicas re-sync on re-connection

grocerylist.js

```
1  var Swarm = require("swarm");
2
3  function initialise() {
4    var Set = require("swarm/lib/Set");
5    groceryList = new Set();
6    groceryList.on(refreshDOM)
7  }
8
9  function addProduct() {
10   var product = document.getElementById("product").value;
11   var quantity = document.getElementById("quantity").value;
12   groceryList.add({p: product, q: quantity})
13 }
```

	Offline Availability	Persisted	Transparent Acces	Consistency
Swarm.js	✓	✓	✓	✗ ✓

✓ Eventual consistency

✗ Limited set of data types

✗ CRDTs require commutative operations

✗ Hard to implement custom ADTs

Conclusion

Manual approach:

Persistent storage:

Network proxies:

Synchronised collections:

Eventual consistency:

	Offline Availability	Persisted	Transparent Acces	Consistency
Manual Caching	✓	✗	✗	✗
HTML5	✓	✓	✗	✗
Service Workers	✓	✓	✗ ✓	✗
METEOR	✓	✓	✓	✗
Swarm.js	✓	✓	✓	✗ ✓