AppNet

A new concept of Internet

Riccardo Del Chiaro

The internet world today

- Web 1.0: Web of Contents - Information
  - Mostly, static web sites with **static information**.
  - Purpose: consultation of information.
  - Eg: www.comune.fi.it

- Web 2.0: Web of Users - Interaction
  - Dynamic web sites, with dynamic information.
  - Apps and WebApps are taking the place of the sites.
  - Content made by users (social).
  - New purpose: **interaction** between humans.
  - New purpose: **interaction** user – service.
  - Eg: www.youtube.com

What users hate of Internet

- Difficulty in finding certain **small information quickly**.
  - "Where can I find the bus timetable under my hotel?"
  - "Where can I find the menu of the day of the school canteen?"
  - "Could I enter this classroom or some professor is lecturing? Where is the class schedule?"

- "**Artificial**" instead of Natural Interactions:
  - To find the right information/site, I have to search by myself.
  - I might be forced to search an information in a maze of links in a site.
  - I might need to write something with the uncomfortable smartphone keyboard only to find a simple information (probably when I'm late!)
  - I might be forced to install a dedicated app for a single information.
    - What if I have the smartphone memory full?
    - What if I'm late? (again) I have to wait app download, installation, start-up, …

- *User is forced to have an active interaction for something that should be simple.*

What we would love

- **Right information** in the **right place**.
  - Right place often means right time.
  - Information should be **simple** and **accessible**.
  - Information could **notify user** when available**.**
- **Natural interactions**:
  - **Minimize number of actions** required to interact with something.
- Minimum **Target**:
  - Automatically find the **physically close things** to interact with.

How to love internet

- **Apps instead of Web Sites**, because:
  - Easy to use, **more naturals** than the classic Web Site approach.
    - Many **sites are becoming Web Apps**.
    - With touch devices, classic Web Site experience is really frustrating.
  - **Touch devices are becoming the most important clients** in the web [1][2].
  - Most of internet users, spends his time in smartphone Apps, not in sites [2].

- More, Few, Physical, Fast
  - **More** dedicated apps.
  - **Few** information in each app/site.
  - **Physical** space as entry point (IoT), instead of URL.
  - **Fast** and flexible access to apps (no install, no storage problems, like sites).

- **Physical** space as entry point**:**

Technology	Active Interaction Level	Confirmation
GPS, Beacon	No	needed
NFC	Small (gesture)	Not needed
QR Code	Medium - High	Not needed

AppNet idea

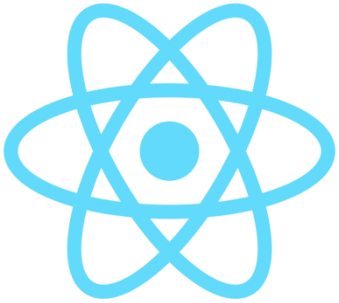
- The dream: a **new standard** for the internet.
- **(Web) Apps instead of Web Sites**.
  - **Localized** with GPS and/or Beacons (or other sensors)
    - Each app have a radius of action.
  - Delivered **on demand**: right app in the right place.
    - User is notified when is in the radius of action of an app.
    - Can receive the app and running instantly without installation.
  - Ideally: **one app for each interactive object/space**.
  - Light and Flexible
    - No install, fast to download, cached in storage automatically to speed up most used.
- **IoT for all**:
  - Objects become **"smart" on the cheap**:
    - Associate WebApp with a GPS location/Beacon that represents a place/object.
    - The logic will be delegated to a server on the internet and/or to the app.
    - GPS: for free, static (good for buildings).
    - Beacon: cheap, dynamic (beacon can move in the world, good for objects and indoor).
  - A more complex object could also be involved:
    - Send WebApp with BT/NFC/Wifi (no need for internet connection. E.g.: TV, fridge, …).
    - Behave as a server, updating in realtime the content of the WebApp with it's own logic.

AppNet: Some Examples

- **Bus**:
  - App for each bus stop: timetables, real-time delay, pay the ticket … (GPS)
  - App in bus: info about next stops, book a bus stop in the path. (Beacon)
- **Train Station**:
  - App for each station: watch timetables and buy tickets (GPS with high radius)
  - App for each binary: delays and arriving trains (GPS low radius or beacons)
- **University:**
  - App for the university building, with general info and schedule (GPS).
  - App for each classroom, with schedule and reservation request (Beacon).
- **Restaurant:**
  - Users can check the app-menu and make order to the table (Beacon).
- **Objects:**
  - Living room: WebApps to control TV, audio amplifier, air conditioner, etc.. each object has it's own WebApp, developed by the producer.
- **People:**
  - Personal app that act like a badge within a school or office (Beacon).

AppNet Technology: ReactNative

- The project has been developed by paying attention to the mobile part (smartphone).



- **WebApp** are written with **ReactNative**:
  - New **promising framework**, developed by Facebook.
  - Write apps in **JS with native GUI** in Android and iOS.
    - No laggy GUI, like other framework totally based on WebViews.
    - Great user experience with minor compromises (only 1 thread).
  - **Share code** between Android and iOS: up to 85% for real world apps [3].
  - Probably: share some code with React apps (run on web browser).

- A lot of **famouse apps are using ReactNative** totally or in part:
  - Examples: Facebook, instagram, Airbnb, Baidu, QQ. [4]
  - It's hard to notice differences in GUI responsiveness with native apps.

- *! For now only the browser for Android has been developed !*

React Native Development

- The base idea of ReactNative
  - Write apps for iOS and Android in JavaScript.
  - Share some code between the two version.
  - Encapsulate the JS app in a single file (index.<platform>.bundle)
  - Native app contain & run the bundle: can be published into the Store.
  - *Users (and OS) will download and run the native app, that will execute the bundle. User can't easily see the difference with a real native app.*

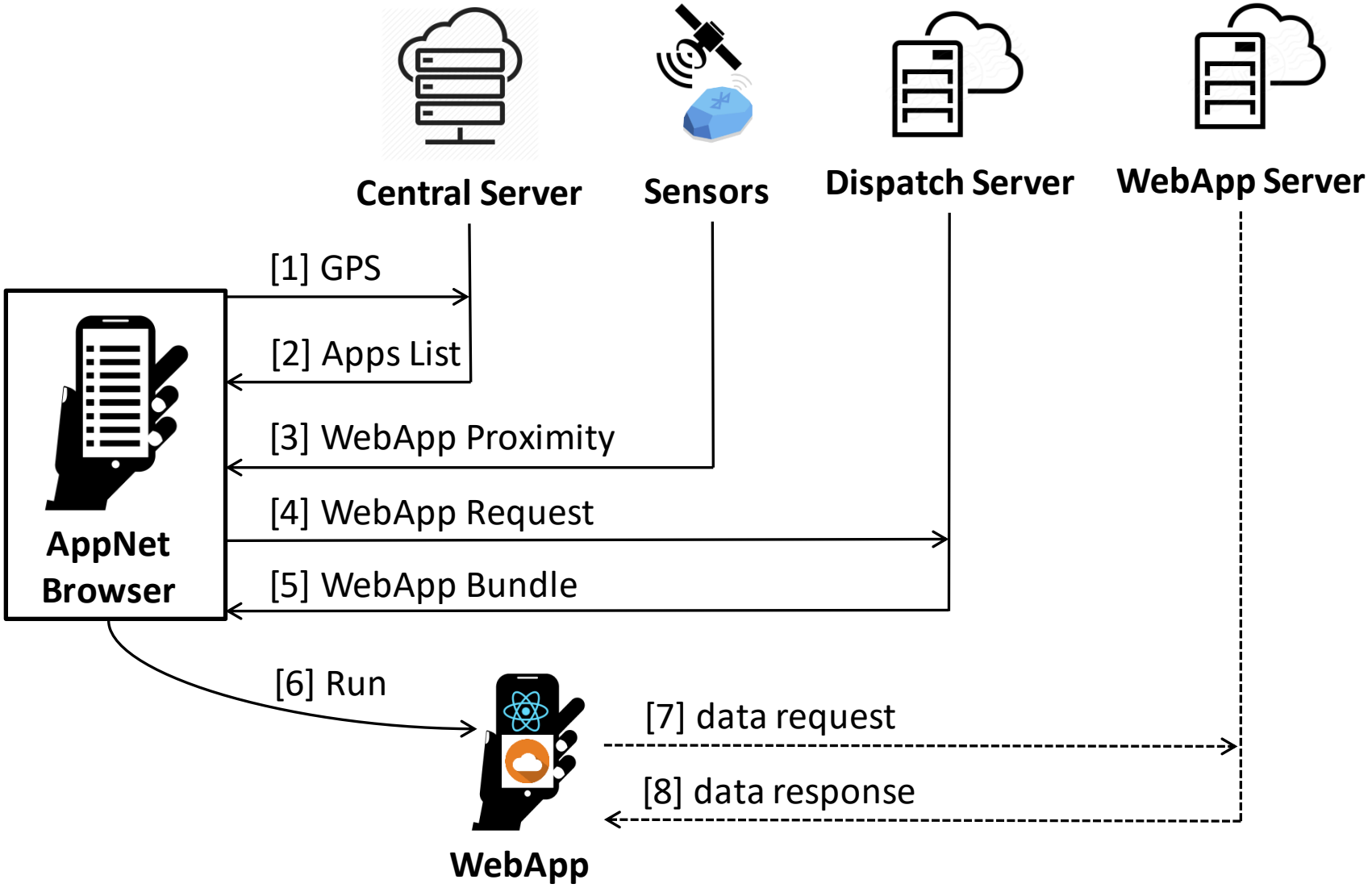
- The AppNet Browser do the same thing, but..
  - …does not contain a bundle…
  - …it downloads a bundle from internet and executes it, on demand.
  - The bundle is like an HTML page for a web browser.

- Each bundle can have user-developed Components.
  - Some introduce new native features: require to recompile the native app (Browser).
  - Others use only JS: can be added without recompile the Browser.
  - A great community on GitHub is developing a ***lot of components*** ***[5]***
    - The browser should integrate the latest native components.

AppNet Structure

- **CentralServer**
  - **Similar to a DNS**: each WebApp will be registered on this server with it's information.
  - For simplicity, acts also as a **DispatchServer**: stores and sends the WebApps to the user.
- **DispatchServer**
  - **Stores one or more WebApp** and sends to the user.
  - Each WebApp could have it's own dispatch server.
  - For simplicity: for now the only DispatchServer is the CentralServer.
- **WebApp Server**: WebApps uses technology similar to WebSites
  - Some information could be stored in a remote server.
  - WebApps could retrieve these information when running (images, video, data).
  - Probably will coincide with the Dispatch Server.
  - NB: this server is not mandatory, is a choice of the WebApp developer.
- AppNet **Browser**
  - An application to browse the AppNet: acts like an internet browser for WebApps.
  - Connects to the central server to discover the WebApps near to the user.
  - **Downloads and run the WebApps** when the user requests to do it.
- AppNet Development Server (Android, iOS)
  - Used to develop new WebApps and test locally before register them in the CentralServer. Now on ReactNativ 0.40.

AppNet Structure



AppNet CentralServer

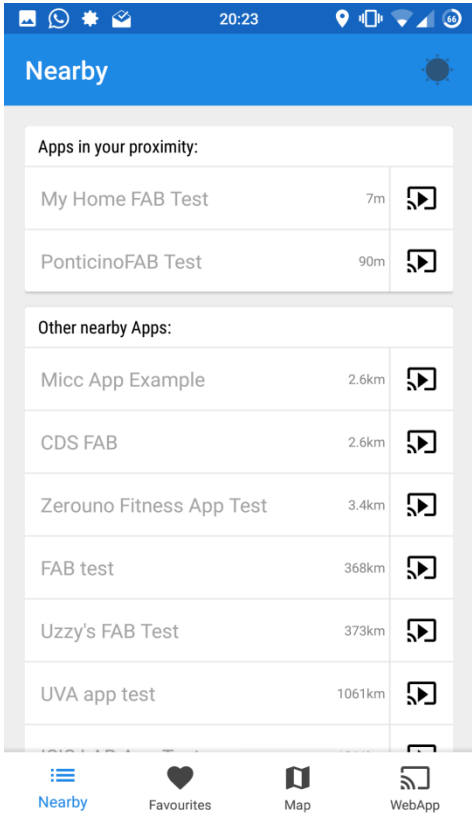
- Made with Python Flask Server.
  - Hosted on C3PO server
- Contains a list of WebApp's metadata
  - Metadata are stored in protobuf binary files (in future should be stored in a DB).
  - When an AppNet Browser connects, server sends metadata of all apps to it.
    - in future should send only the metadata of apps close enough to the user (easy with a DBMS).
- Web interface to manage, modify and insert new WebApps
  - Because for now the CentralServer is the only one DispatchServer, we have to upload the WebApps through this web interface.
  - We can also associate a GPS location and a Beacon to a WebApps through this interface.

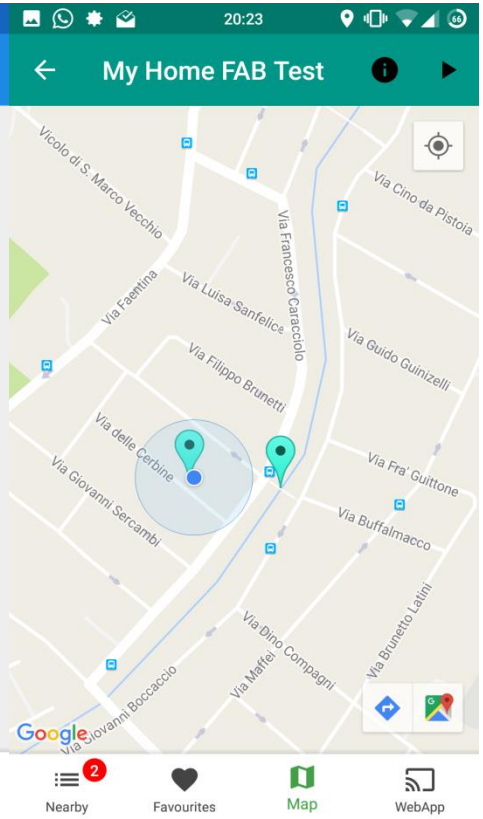
AppNet Development

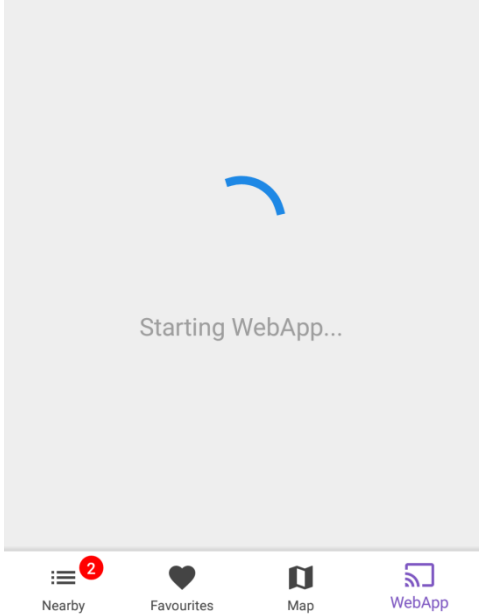
- ReactNative Project with the native libraries synchronized with those used by the Browser.
- Each ReactNative Project contains
  - Android and iOS native projects:
    - Compile and install on the Device to connect to the debug server.
  - A reactNative debug server (packager)
    - Packs the JS application: makes a bundle to be send to the debug client.
  - The Javascript Application:
    - Index.android.js: android entry point.
    - Index.ios.js: iOS entry point.
  - Folder node_modles:
    - Contains all the ReactNative modules/libraries of the project.
    - Native modules: must be the same used by the browser.
    - JS-only modules: may be different from those used by the browser.

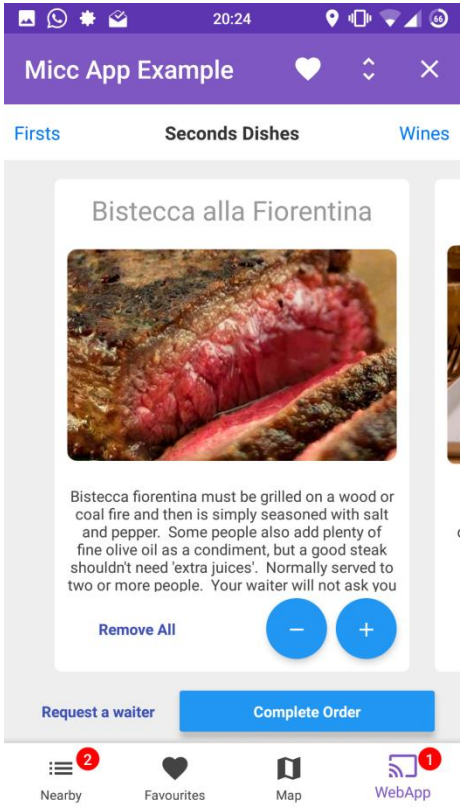
AppNet Browser

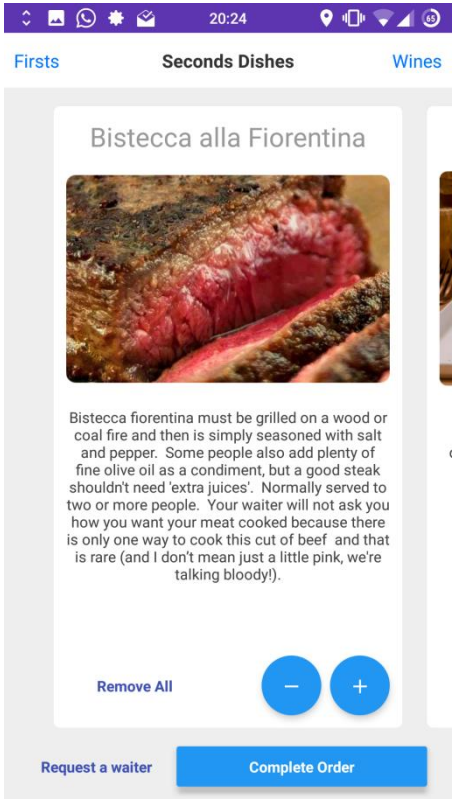
- Native app developed for Android (Java).
- Browses the WebApps that are in the CentralServer.
  - Sorted by distance.
  - The webApps close enough will be in evidence (Proxy List)
  - WebApps associated to the beacons in the proximities, will be in evidence (Proxy List).
- Looking at the ProxyList, user can easily access to most important information nearby.
- A Floating Action Button will appear when a beacon is in proximity.
  - It's an easy access for the closest beacon.
  - A sort of "I'm feeling lucky"
- User can also browse the GPS-Localized apps in a MapView.


20:23
Nearby
Apps in your proximity:
My Home FAB Test 7m
PonticinoFAB Test 90m
Other nearby Apps:
Micc App Example 2.6km
CDS FAB 2.6km
Zerouno Fitness App Test 3.4km
FAB test 368km
Uzzy's FAB Test 373km
UVA app test 1061km
Nearby
Favourites
Map
WebApp


20:23
My Home FAB Test
Vicolo di S. Marco Vecchio
Via Cino da Pistoia
Via Francesco Caracciolo
Via Faentina
Via Luisa Sanfelice
Via Guido Guinizelli
Via Filippo Brunetti
Via delle Cerbine
Via Giovanni Sercambi
Via Fra' Guittone
Via Buffalmacco
Via Dino Compagni
Via Brunetto Latini
Via Giovanni Boccaccio
Google
Nearby
Favourites
Map
WebApp


Starting WebApp...
Nearby
Favourites
Map
WebApp


20:24
Micc App Example
Firsts
Seconds Dishes
Wines
Bistecca alla Fiorentina
Bistecca fiorentina must be grilled on a wood or coal fire and then is simply seasoned with salt and pepper. Some people also add plenty of fine olive oil as a condiment, but a good steak shouldn't need 'extra juices'. Normally served to two or more people. Your waiter will not ask you
Remove All
Request a waiter
Complete Order
Nearby
Favourites
Map
WebApp


20:24
Firsts
Seconds Dishes
Wines
Bistecca alla Fiorentina
Bistecca fiorentina must be grilled on a wood or coal fire and then is simply seasoned with salt and pepper. Some people also add plenty of fine olive oil as a condiment, but a good steak shouldn't need 'extra juices'. Normally served to two or more people. Your waiter will not ask you how you want your meat cooked because there is only one way to cook this cut of beef and that is rare (and I don't mean just a little pink, we're talking bloody!).
Remove All
Request a waiter
Complete Order

AppNet Browser: Implementation

- The code is organized in 2 macro-packages:

- **Model**: contains all the packages and classes that manage the functionalities of the browser.

- **Controller**: contains all the packages and classes that manage the GUI and present data.

The next two slides will show a view of these two packages, with a short explanation of the purpose of the packages and of the most important classes.

AppNet Browser: Model

- **Package connection**: manages the connection with the CentralServer and the download of the AppList and of WebApps (bundle).
- **Package beacon:** manages the beacon functionality through the Estimote Library.
  - **EddystoneScanner:** scans for beacons every N seconds, and notifies listeners at each scan for connected beacon, new connected beacons and lost beacons.
  - **EddystoneProximity**: uses EddystoneScanner to make geofencing with beacons. Notifies the listeners for geofence events on beacons (entering, exiting, in, out).
- **Pakage localization:** helper package that use Google API to get FusedLocation updates; using GPS, mobile network and wifi.
- **Package geofencing:** package that manages the GPS geofence. Implemented from scratch, doesn't use the Google Geofence API because of the limitations.
  - GeofenceManager: the entry point of the package, executes a GeofenceScanTask following some interval rule (timer, manual, GPS updates, …)
  - Geofenceable: the objects to which apply the geofencing must implements this interface.
- **Package webapp:**
  - Manage the **FavoruiteApps**.
  - Define the **WebApp** class (Parcelable and Geofenceable) and WebAppList class.
    - Insatiable from the protobuf objects.

AppNet Browser: Controller

- **MainActivity**: the application main activity, central point of application.
- **MainFragment**: an extension of the Android Fragment that implements some helper methods to interact with the MainActivity.
- **Package fragments**: package that contains the 4 MainFragments of the project (NearbyList, FavouritesList, Map, WebAppContainer).
  - **ReactFragment**: manages and runs the ReactNative WebApplication.
    - Managed by WebAppContainerFragment.
    - ReactNative's "native" libraries (java), have to be loaded in the method ReactFragment#onAttach(…), through the line builder.addPackage(..)
- **FragmentHelper**: a generic helper class to simplify the swap of generic fragments.
- **FragmentController**: uses the FragmentHelper and manages the swap of the 4 MainFragments of this project.
- **WebAppController**: notifies the other controllers when the model's state (webApp lists) is changed.

Conclusions

- The initial idea imply a really big work
  - The work done want to be a sketch of the main concept.
- It shows that the idea can be implemented, works and could have a future
  - All the initial technological problems were overcome.
- Main target has been achieved, key features have been implemented:
  - Right WebApp/Information in the right place.
  - On-demand apps that behave like native apps.
  - No invasive app: light weight and no installation.
  - Minimized the active actions that an user have to make for interact with apps/info on objects and places nearby.

Future Developments

- Browser for other platforms: iOS, Windows/Linux/OS X (browser based, React).
- Support iBeacon Technology.
- Separate CentralServer from DispatchServer.
- Favourites organization by distance and by name with a folder/tag system.
- Making a solid server-side website for developers
  - Decent GUI to submit new apps and DispatchServers IP:PORT, with Authentication System.

- Making an (optional) authentication systems for WebApp Browser users:
  - same profile to use with all WebApps, each WebApp could use user's profile information if agree.

- Customization of WebApp geofencing radius from the Metadata stored in CentralServer (both for Beacons and GPS).

- Customization of geofencing 'extra-radius' by the browser user.
  - An user might want to be advertised also when is It is a bit more distant.

- Implement the concept of 'skimming-radius'
  - Browser should request to the CentralServer only WebApps localized within some kilometers from the user position.

Future Developments

- For WebApps in mobility (no GPS, only beacons), a ‘last-spotted’ system could be implemented
  - Estimate the position of a beacon using the last sightings from the browsers: browser’s GPS position as an estimation of beacon position.
- Implement a Machine Learning technique to predict the interest of the user
  - Focus the attention of the user on some WebApps, also if they are not in proximity.
  - Could predict that the user will be soon nearby, for some reason (Eg: usual home-office path).
  - Could notify the user for a WebApps of particular interest, with system notifications.
- Personalizable templates (WebApps - WebApp Server), so that non-expert users can make personal WebApps without high technical skills
  - Restaurants, public buildings, transport timetables, shops, arduino and raspberry (Domotica), personal social profile on beacon, …
- Implementation of a BeaconService to turn a smartphone or a PC into a beacon.
  - E.g.: PC dispatches a WebApp for remote control, file exchange, …
- Consider the implementation in the WebApp Browser of modules/plugin to run WebApps made with other technologies:
  - Angular, Polymer, Polymer Fire, and others JS frameworks.
  - Could be counterproductive if offers a worse UI experience (without native components).

References

- [1] http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/ - mobile marketing statistics
- [2] http://marketingland.com/digital-growth-now-coming-mobile-usage-comscore-171505 - internet usage
- [3] http://nerds.airbnb.com/facebook-react-native/ - 85% of code sharing between iOS/android.
- [4] https://facebook.github.io/react-native/showcase.html - famous apps using ReactNative.
- [5] https://facebook.github.io/react-native/releases/next/docs/more-resources.html - ReactNative Community