Firefox OS Overview

Ewa Janczukowicz

Collection des rapports de recherche de
Télécom Bretagne

RR-2013-04-RSM

Firefox OS Overview

Ewa JANCZUKOWICZ (Télécom Bretagne)

TELECOM Bretagne
Institut Mines-Télécom

This work is part of the project "Étude des APIs Mozilla Firefox OS" supported by Orange Labs / TC PASS (CRE API MOZILLA FIREFOX OS - CTNG13025)

ACKNOWLEGMENTS

Above all, I would like to thank Ahmed Bouabdallah and Arnaud Braud for their assistance, support and guidance throughout the contract.

I am very grateful to Gaël Fromentoux and Stéphane Tuffin for giving me the possibility of working on the Firefox OS project.

I would like to show my gratitude to Jean-Marie Bonnin, to all members of Orange NCA/ARC team and RSM department for their help and guidance.

SUMMARY

Firefox OS is an operating system for mobile devices such as smartphones and tablets. It is developed by Mozilla but it aims to be free from any proprietary technology. It lets users run applications developed entirely using web technologies, like HTML5, JavaScript, CSS.

It is not directly competing with iOS. It has some common target markets with Android. For now Firefox OS targets specific type of clients - people that don't have smartphones yet. Available devices are low-end and as a result prices of Firefox OS phones are low.

Firefox OS is aimed to be free from a proprietary technology so, as an effect, flexible and open. There should be no more device or vendor fragmentation. All apps are supposed to be built once and run everywhere. It is possible thanks to Web Apps and Web APIs. Mozilla makes a big effort in order to standardize Web APIs, so that the device hardware could be accessed more easily.

Mozilla also introduced its sign-in system for the Web. Mozilla Persona allows sign-in by using any of user's existing email addresses. Persona implements a BrowserID protocol that is a universal login system that does not require email providers to support it. Furthermore, the identity provider cannot track user's activity thanks to a certification system. Implementing persona requires very little code so it can be quickly deployed. At this point the biggest issue that Mozilla is facing is the lack of websites using their system.

Mozilla also wants to redefine the way payments work for mobile applications. Firefox OS allows two types of paid content: paid applications and in-app payments. There are three important parts in the payment process: Payment Provider, Client App and App Server. They communicate with each other and use a payment token to provide all necessary information concerning the product being purchased. Mozpay, the principal part of the payment flow, was introduced in the first version of Firefox OS. Recently mozpay was proposed to be depreciated, because it made the payment process to rigid for certain payment providers. There are several solutions like PayPal, Google Wallet or Stripe that can be easily added to the web content by injecting JavaScript into web pages. The new solution is to securely expose Payment Provider primitives that websites can use to implement mobile payments. The subject is still under discussion.

Firefox OS is still in the early phase. Different features and functionalities keep changing. The existing applications are not all working smoothly yet but the marketplace keeps growing.

So far Firefox OS was launched in several countries like Spain and Poland. There are other carriers that will start selling it in other countries sometime soon.

The opinions about Firefox OS are divided. So far all sold devices are low-end so they are targeting a certain type of clients, users that are not very demanding. As a result it is difficult to clearly evaluate this system or to compare it to existing technologies. The future of Firefox OS is still unsure. Although thanks to growing interest in Web Apps and Web APIs and cooperating with other companies and mobile carrier the system may become successful.

KEYWORDS : API, Web app, Firefox, OS, payment, mozpay, identity, Persona

Table of contents

Table of figures

Abbreviations

API	Application Programming Interface
App	Application software
CSS	Cascading Style Sheets
HAL	Hardware Abstraction Layer
IdP	Identity Provider
JSON	JavaScript Object Notation
JWT	JSON Web Token
OEM	Original Equipment Manufacturer
OS	Operating System
RP	Relying Party
SDK	Software Development Kit

1. INTRODUCTION

Firefox OS is an operating system for mobile devices such as smartphones and tablets. It is developed by Mozilla but it aims to be free from any proprietary technology.

The aim of this state of the art report is to describe the Firefox OS environment. The research is mainly focused on Web Apps and Web APIs but also on Mozilla's solutions concerning identity and payment.

In the first section, the general Firefox OS environment is presented. Firstly, the characteristics and motivations of target users are given. Secondly, operating systems, which can compete with Firefox OS, are described. Finally, architecture and design of Firefox OS are presented.

The second section is devoted to Web Apps and Web APIs. Web Apps are applications that are built using web technologies, like HTML5, JavaScript and CSS. Web APIs allow accessing devices hardware and data stored on a given device. Thanks to Web APIs, Web Apps can interact with the hardware by using JavaScript. Firstly, the history of Web Apps is presented. This chapter is followed by a comparison of Web Apps and native applications. Secondly, more details of different types of Web Apps are given. The process of creating and distributing these applications is presented. Finally, different types of Web APIs and corresponding security policies are listed.

The third section is focused on the identity solution introduced by Mozilla. Mozilla Persona (previously known as BrowserID) is an alternative to Google Login or Facebook Connect. It allows signing in by using any of user's existing email addresses. Firstly, the sign-in process is presented. Secondly, the BrowserID protocol used by Persona is described in detail.

The fourth section is devoted to the payment solution. Firstly, Mozilla's strategy and business model are given. Secondly, paid apps with a receipt protocol are described. Thirdly, in-app payments and mozpay technology are presented in detail, including the whole payment flow. The research presented in this chapter is valid for Firefox OS v.1.0. Mozilla's payment model is still a work in progress and may completely change in the future. For now Mozilla is probably going to depreciate mozpay and focus more on Payment Provider primitives.

In the end of the report the conclusions are given. The strengths of the system are presented, as well as ongoing improvements and launches in different countries.

2. FIREFOX OS ENVIRONMENT

Firefox OS [1] is an operating system for mobile devices such as smartphones and tablets. It is developed by Mozilla but it aims to be free from any proprietary technology.

Firefox OS uses a Linux kernel and boots into a runtime engine that lets users run applications developed entirely using web technologies, like HTML5, JavaScript, CSS. Thanks to this solution, the entire user interface is a web application that is capable of launching other web apps, which are just web pages with enhanced services and access to device's hardware.

2.1 TARGET USERS

The founders of Firefox OS want it to be one of the three most popular operating systems. It is not directly competing with Android or iOS, although it has some common target markets with Android. Firefox OS targets mostly emerging countries and does not plan to launch a high-end offer.

2.1.1 Developers

Creators of Firefox OS understand that no mobile operating system can exist without a rich marketplace. In order to attract new users, marketplace needs to grow fast. That is why Mozilla wants to encourage developers to create web apps by offering them various advantages, like: simplicity, open standards, developed community.

Building web apps for Firefox OS is simple, because only web technologies are used. Web apps are flexible and can be run across different platforms. Also Mozilla is working on a set of open standards, for example for accessing device's hardware, that will even more simplify the development process and will make web apps more universal. Firefox OS is not limited to one market place so there is no vendor controlled ecosystem. Developers may upload their apps to any marketplace or just distribute it from their websites. Firefox OS is also attractive because Mozilla has already an existing community of about 450 million of Firefox's desktop users.

Web technologies are easy to get familiar with for developers. In case of developing a simple web application there is even no SDK needed. It can help to grow the marketplace fast but it may also lead to a lot of low quality apps, built by semi-developers.

Mozilla cooperates with Geeksphone that offers two phones created especially for developers (building apps, testing, etc.):

- Keon (91€),
- Peak (149€).

Prices are not elevated although it is not easy to purchase those phones, since there are a lot of developers interested in them.

2.1.2 Clients

As it was said before, Firefox OS was created to become one of the three most popular mobile operating systems, although it does not compete directly with Android or iOS. It targets different type of clients - people that don't have smartphones yet. That is why Mozilla's operating system is offered on low cost phones with a very basic hardware.

Mozilla is cooperating with different manufacturers and carriers that help them to launch Firefox OS based phones. Cooperating manufacturers are: Alcatel, ZTE, Geeksphone, LG, Huawei, Sony. Main cooperating carriers are (distributing Firefox OS with their mobile contracts): Deutsche Telekom, Sprint and Telefonica. At first Mozilla launched Firefox OS phones in two countries: Spain and Poland. In Spain clients of Telefonica's low cost line (Movistar) could buy a ZTE Open device (69€ with 30€ of credit included) since July 2013. Also in July 2013 T-Mobile Poland offered the Alcatel One Touch Fire for 99€ or 0.23€ with a mobile contract (about 10€/month). Deutsche Telecom already sales Firefox OS devices in Germany and is planning to launch Firefox OS phones in two other countries: Hungary (Magyar Telekom) and Greece (Cosmote).

So far all Firefox OS phones are low cost. The hardware and performances are limited, although it may be enough since target clients are users that purchase this phone as their first smartphone ever. They don't have a lot of expectations and they are not comparing the performances with high end devices.

2.2 COMPETITION

There are two operating systems that have a very strong presence on the market: iOS and Android. There is also Windows Phone that is trying to build its way. There is already quite a competition and except Firefox OS, there are other systems that are already under development. For this moment Firefox OS is the only one that was officially launched and that has already gained a certain amount of users. Being the "first" on the market is its big advantage.

2.2.1 Tizen

Tizen [2] is an open source Linux based mobile OS created by Samsung and Intel. Cooperating manufacturers are: Samsung, Fujitsu, Huawei and mobile carriers are: NTT Docomo, Orange, Vodafone and SK Telecom. This OS offers a hybrid approach, which means using native application layer along with web technologies such as HTML5. At some point there were some rumors that the OS was abandoned but in July they launched Tizen App Challenge with over $4M of prizes for the best apps and future OS releases were announced.

2.2.2 Ubuntu Touch

Ubuntu Touch [3] is an open source mobile interface for Ubuntu created by Canonical. It also offers a hybrid approach: native applications (built in Qt) and web technologies (HTML5). It uses the same technologies as Ubuntu Desktop what makes it different is that it provides a desktop experience when connected to an external monitor.

Summer 2013 Canonical launched a crowd-funding project in order to collect money and build a smartphone dedicated to Ubuntu Touch. Even though it collected a significant amount of money, it was below their expectations. Ubuntu still decided to launch its mobile OS for supported devices in October 2013. Supported devices are Galaxy Nexus, Nexus 4, 7 and 10.

2.2.3 Sailfish

Sailfish [4] is a Linux based mobile OS with a proprietary user interface. It is created by Jolla (ex-developers of Nokia's MeeGo). Cooperating carrier or manufacturers are not confirmed yet but the system can be run on Android hardware. It offers a rich hybrid technology and is available to run Android apps in addition to HTML5 and Qt. Jolla smartphones are currently available for pre-order in Finland.

2.3 ARCHITECTURE AND DESIGN

2.3.1 Architecture

Firefox OS is based completely on web technologies. The whole operating system is kind of a web browser that is able to launch other web applications.

There are three main layers:

- Gaia,
- Gecko,
- Gonk.

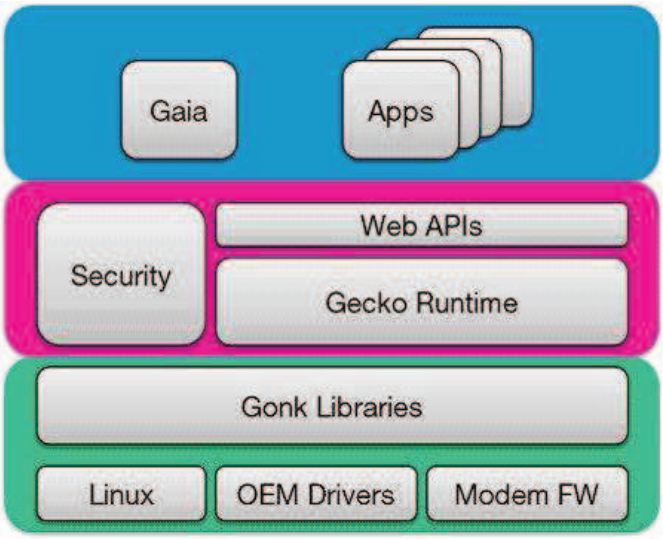

Figure 1: Firefox OS Architecture [5]

Gaia

Gaia [6] is a user interface, so everything that appears on the screen after Firefox OS starts. Technologies used to develop Gaia are: JavaScript, HTML and CSS. It accesses lower levels by using Web APIs. As long as standard APIs are used, it is not important what kind of operating system is used below.

Gecko

Gecko [7] is the application runtime. It is the layer that provides framework for app execution. It implements HTML5, CSS and JavaScript, so its function is to read the web content and render it to user's screen. It also implements Web APIs used to access device's hardware.

Gonk

Gonk [8] is the lower lever operating system. It is just a very simple Linux distribution. It consists of a Linux Kernel, software libraries and hardware abstraction layer (HAL). Gecko can be run on other operating systems, although Mozilla has a full control over Gonk, so more interfaces can be exposed to Gecko.

2.3.2 Builds

Firefox OS (or B2G as it was called before) can be built and installed on different mobile devices. There are some devices designed especially for Firefox OS. It can be also built on Linux or Mac and flashed to other supported devices, but this action will remove operating system currently installed. Unfortunately performances on devices that are not dedicated to Firefox OS are not perfect, for example:

- Samsung Galaxy SII - system runs very slowly;
- Samsung Galaxy Nexus – the phone does not have a home button, so it is not possible to go back to the main screen;
- Samsung Nexus S – gives the best results although the size of the home screen is reduced.

It is also possible to install Firefox OS as a desktop client, although no device APIs will be accessible. The fastest way to test a web app is to use a Firefox OS simulator that can be installed within Firefox browser.

2.3.3 Design

The design of Firefox OS is not very different from interface that is known in Android or iOS.

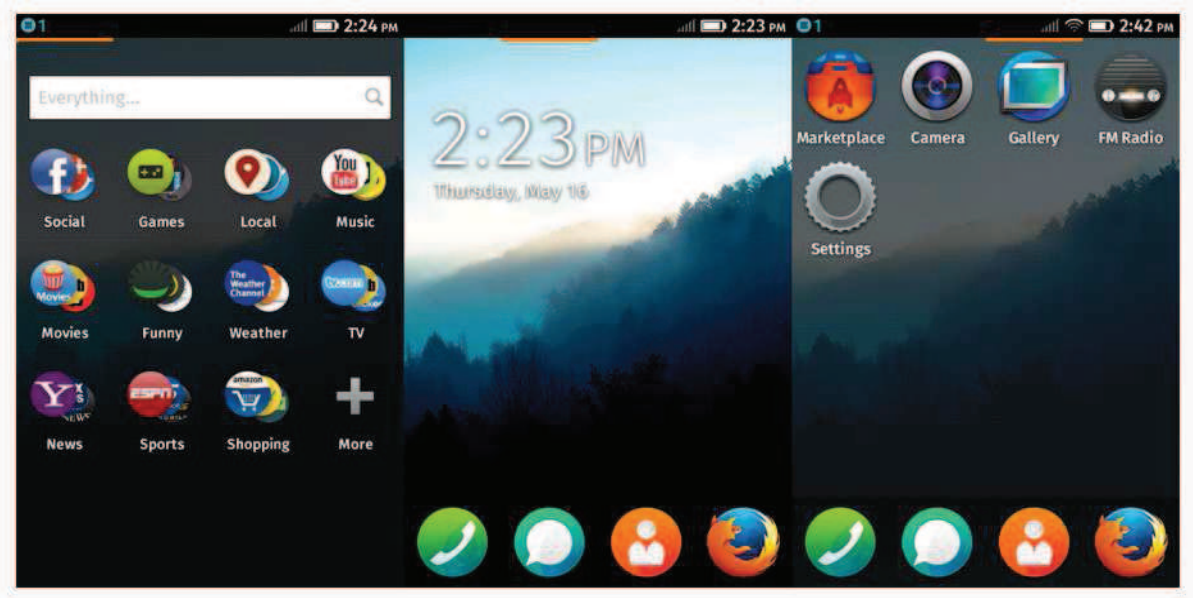

Figure 2: Firefox OS Screen Shots [9]

There is a typical home screen (in the middle) and the notifications/settings can be seen here when sliding down. When sliding to the right, all installed applications can be seen. When sliding to the left, the research window is shown. This allows searching for an application; either installed locally either on the Internet.

3. WEB APPS AND WEB APIS

Firefox OS is aimed to be free from a proprietary technology so as an effect flexible and open. There should be no more device or vendor fragmentation. There is no need to develop apps for a specific hardware or operating system. All apps are supposed to be built once and run everywhere. It is possible thanks to Web Apps and Web APIs.

Web Apps are applications that are built using web technologies, like HTML5, JavaScript and CSS. They can run on any modern web browser, including those for desktops and mobiles devices [10].

Web APIs allow accessing devices hardware and data stored on this device. Thanks to them it is possible develop web applications that interact with the hardware by using JavaScript. Mozilla is trying to make those APIs standard so there would be consistent APIs for all browsers no matter the operating system or device. [11]

3.1 FOCUS ON WEB APPS

3.1.1 History of Web apps

The story of Web apps started in 2007. Apple launched its iPhone and wanted to have a full control over their technology. Apple didn't want to allow any third party native apps because they could cause devices crashes and as an effect ruin iPhone's reputation. That is why the company thought about Web apps that were defined as little programs that run inside the mobile browser.

Steve Jobs was enthusiastic about the idea: *“The full Safari engine is inside of iPhone. And so, you can write amazing Web 2.0 and Ajax apps that look exactly and behave exactly like apps on the iPhone. And these apps can integrate perfectly with iPhone services. They can make a call, they can send an email, they can look up a location on Google Maps.*

And guess what? There’s no SDK that you need! You’ve got everything you need if you know how to write apps using the most modern web standards to write amazing apps for the iPhone today. So developers, we think we’ve got a very sweet story for you. You can begin building your iPhone apps today.” [12]

The result was not what Steve Jobs expected. The browser technology was not there yet. Built web apps didn’t give enough control over the phone and needed to access wireless network all the time. Users started “jailbreaking” their phones in order to install not official native apps that could be installed directly on the phone and run faster than Web apps. In order to discourage users from using unofficial applications, in 2008 Apple announced an official SDK for native apps and an App Store opening for business. As a result ever since Apple has a full control over all applications and a significant revenue share.

There were other attempts to use HTML5 to build different applications. Wooga a social network game and mobile developer tried to develop its game “Magic Land Island” in HTML5. They started working on it in 2011 but in June 2012 they stopped the development and released it as an open source project. The project was abandoned because the browser technology was not advanced enough and the app was not running smoothly, for example there were problems with updating and charging different levels. Although Philipp Moeser, co-founder and CTO at Wooga still believes in HTML technology: *“We’re very proud of the work we’ve done with HTML5 over the past year. With some of the most talented software engineers in the industry working on the project here at Wooga, we’re confident that the community will find lots to learn within Pocket Island and use our experience to progress the technology even further. HTML5 certainly has the potential to be a complete game changer, but the technology isn’t there yet”.* [13]

Also Facebook abandoned the project of creating a Facebook app in HTML, because the app was to slow and not responsive enough. Mark Zuckerberg, the founder of Facebook became less optimistic about HTML5 technology: *"We burnt two years, it was really painful. I think probably we'll look back and say it was one of the biggest mistakes, if not the biggest strategic mistake we've made. But, we're coming out of that now."* [14]

There are also applications that prove that HTML5 and Web apps are worth investing in. An award-winning Financial Times Web app proves that web apps can be as good as native apps. Normally all apps that provide magazine subscriptions have to not only share their revenues with app stores but also let the app stores maintain all the data. Financial Time wanted to maintain total control over the subscription pricing and subscriber data, so it decided to build a Web app that could be saved on a phone as a bookmark. The app was launched in 2011 and after a year over 1 million users switched to it from the native app. Mary Beth Christie, Product Director even said that most of the users don’t see any performance difference: *“We’ve had super positive response from users overall. But the experience is so close many people probably did not recognize the difference”.* [15]

3.1.2 Web apps vs. native apps

Given the history, in some cases Web apps work well, but for certain types of applications the technology is not advanced enough. It is not possible to compare native apps to Web apps, because it is like comparing apples and oranges [16]. Both types of apps have the same usage, although they behave in a completely different way. Native apps have better performances: they are faster and more responsive. On the other hand they are not flexible at all. They seem to be tied to the hardware and business models. Web apps are the opposite, they can be written once and deployed everywhere but it has its price: they are less optimized and slower.

There also exists a hybrid approach, for example by using a Phone Gap. Web apps can be encapsulated and run on different platforms. Of course this approach is also limited and influences the performances. PhoneGap [33] is an open source framework. It uses to bridge web applications and mobile devices. Thanks to this approach it is possible to build cross-platform applications by using HTML5, JavaScript and CSS.

3.1.3 Types of applications/web content

There are two types of Firefox OS applications: hosted and packaged. Hosted apps are those that are run from a server at a given domain. Packaged apps are zip files containing all app assets (HTML, CSS, JavaScript files and app manifest).

Hosted apps

Hosted apps can be run or installed from any website. It proves the fact that Firefox OS was designed to be open and free from any proprietary technology, so there is no app store and no restricted business model needed.

Hosted apps can be divided into two types:

- Normal web content – normal websites that can be displayed in a mobile web browser. They cannot access to critical or sensitive functionalities of a device and all explicit permissions need to be requested at a runtime.
- Installed Web apps – they are pretty much the same thing as a normal web content. The main difference is that they can be installed on a device. Since no app store is used, there are no code reviews. Apps are installed from a website and there can be only one app by origin. All explicit permissions should be requested at runtime and the data usage intensions should be showed to the user.

Packaged apps

Packaged apps are zip files containing all apps assets like HTML, CSS and JavaScript files. At first Mozilla wanted to implement only hosted apps but finally for security reasons they decided to implement packaged apps. It was the only way to sign applications and assure secure access to sensitive APIs.

Depending on security and access levels there are three types of packaged apps:

- Regular – it corresponds to hosted installed Web app. The only difference is that it is packed in a zip file and can be downloaded from an app store. It does not have access to any critical or sensitive functionalities of a device and all explicit permissions and data usage intensions have to be requested at runtime. It can be signed by marketplace but without any advanced code review or authentication process.

- Privileged – applications that are equivalent to a native app that can access sensitive functionalities of a device. For security reasons they require authentication and code review by a trusted store that also signs them cryptographically. All explicit permissions are requested at runtime and data usage intensions are showed to the user. Also Content Security Policy for preventing cross-site scripting attacks needs to be used.
- Certified – applications that can access critical functionalities of a device. It needs a high security level so an approval of carrier or OEM is necessary. All permissions are implicit and cannot be modified by a user. As for privileged apps Content Security Policy for preventing cross-site scripting attacks needs to be used.

3.1.4 App manifest

When creating Web apps for Firefox OS, the important difference between a Web app and normal web content is that a Web app needs to have an app manifest. An app manifest is a JSON file providing important information about the app: version, name, description, and domains the app can be installed from. What is the most important it contains a list of Web APIs that an app wants to access with a description why a given privilege is needed – that allows user to consciously install an app. It needs to be hosted at the same origin as the app. Also there can be only one app per origin, because otherwise apps would be able to examine each other's local storage. The origin of an app is the protocol, domain and port of the URL – all together. [34] gives some examples:

- Each of the following URLs is a different origin:
  - http://example.com
  - http://example.com:8080 (different port)
  - https://example.com (different protocol)
- The following URLs are the same origin:
  - http://Example.com:80
  - http://example.com
  - http://example.com/drawingApp
  - http://example.com/notesApp

App manifest after the installation is stored locally, so the device can check for unexpected changes in case for example application updates.

3.1.5 Creating a Web app

On the Figure 3 a flow of creating a Web app is presented.

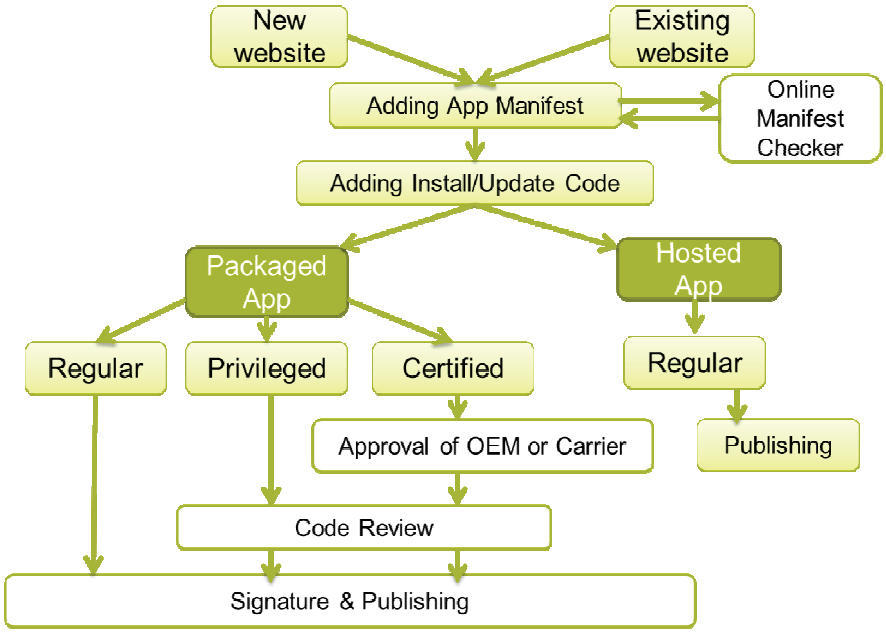

Figure 3: Web app creation

Firstly a developer may either create a completely new website or use an existing one. Later an app manifest for this app needs to be created. It can be checked via an online manifest checker [17]. What makes a Web app different from a traditional website is that it can be installable – it is achieved by adding an Install/Update code to a website.

Secondly a developer should decide on what Web APIs he wants to use.

- If he uses only regular APIs he can either build a packaged app or a hosted app. If he decides to create a hosted app he can directly deploy it on his server or any publicly accessible Web server, for example GitHub. If he decides to build a packaged app he can post in on a Marketplace, it will be signed but no detailed code review will be performed.
- If he uses privileged APIs, he can only create a packaged app that is equivalent to a native app. The app will have to pass a code review (done by a trusted marketplace) before being signed and published on a marketplace.
- If he uses certified APIs, he needs to get an approval of OEM or Carrier. Certified apps are used for critical functions on a Firefox OS device. They also need to pass a detailed code review before being signed and published - usually they are preinstalled before they get to the final user.

3.1.6 Distribution channels

Mozilla does not lock its users (clients or developers) into one marketplace or business models. It gives the possibility of creating different marketplaces. It is also possible to access or install Web apps directly from a website.

The Figure 4 shows all possible distribution channels and the Web apps that they provide. It is shown that certified apps can be only preinstalled on the phone. It is done for the security reasons since they access critical services. However for security reasons privileged apps can be only installed from Mozilla approved marketplaces (Firefox OS Marketplace or Mozilla's partners' stores).

Distribution channel	Certified	Privileged	Regular
Preinstalled on a Phone	✔	✔	✔
Official Mozilla Marketplace - Firefox OS - Firefox Browser - Firefox Aurora for Android	✗	✔	✔
3rd Party Marketplace Approved by Mozilla	✗	✔	✔
3rd Party Marketplace Not Approved	✗	✗	✔
Website	✗	✗	✔

Figure 4: Web apps distribution channels

Currently there are over 2000 apps on the Firefox Marketplace but this number is increasing every day.

3.2 Focus on Web APIs

Mozilla makes a big effort in order to standardize Web APIs. Thanks to those APIs, mobile applications would no longer be built for specific devices but for HTML5 in general. As a result, apps will be built once and run everywhere. There will be no more device fragmentation. HTML5 application will be able to access hardware or data storage directly by using JavaScript.

3.2.1 Types of Web APIs

As it was mentioned in previous chapters there are different types of Web APIs with corresponding security levels.

Regular APIs can be accessed without any restrictions. Privileged APIs are reserved for packaged Web apps and require a code review and signature by a trusted app store. Certified apps are reserved for packaged Web apps and require an approval of OEM or carrier, a code review and a signature – those are apps that need to be preinstalled on a device.

The Figure 5 shows all available APIs divided into four categories. The last column (future APIs) represents APIs that are not fully developed yet so the security and access level for them has not been defined so far. All APIs that are already under standardization are marked with '*'.

Regular APIs	Privileged APIs	Certified APIs	Future APIs
Vibration API* Screen Orientation* Geolocation API* Mouse Lock API* Open WebApps* Network Information API* Battery Status API* Alarm API* Web Activities Push Notifications API* WebFM API WebPayment IndexedDB* Ambient light sensor* Proximity sensor* FMRadio	Device Storage API Browser API TCP Socket API* Contacts API* systemXHR	WebTelephony* WebSMS* Idle API Settings API Power Management API Mobile Connection API WiFi Information API WebBluetooth Permissions API Network Stats API Camera API Time/Clock API Attention screen Voicemail	Resource lock API UDP Datagram Socket API Peer to Peer API* WebNFC WebUSB HTTP-cache API Calendar API Spellcheck API LogAPI Keyboard/IME API WebRTC FileHandle API Sync API

Figure 5: Types of Web APIs

3.2.2 Security policies

As it was explained in the chapter before, there are different security policies. Permissions may be explicitly or implicitly granted.

Explicitly granted permissions are for regular and privileged APIs. They are enumerated and described in the manifest and require a user consent at a runtime. They can be modified via the permission manager (that is a big advantage comparing to native apps for which changing permissions in more complicated).

Implicitly granted permissions are reserved for certified APIs. They are also enumerated in the manifest although they are granted without even prompting the user. User can inspect them but he cannot modify them.

3.2.3 Available Web APIs

WebAPI wiki [18] gives a list of all available APIs. APIs names are quite self-explanatory. Most of them allow accessing devices hardware (Vibration API, Battery Status API), devices storage (Device Storage API) or network functionalities (Mobile Connection API, TCP Socket API).

There are some APIs that give developers interesting features. For example Push Notifications allow apps to be woken up when receiving notifications [19]. Mozilla also implements WebActivities that are similar to Google's Web Intents. WebActivities allow inter-app communication and delegating an activity to another application [20].

As it is shown in wiki [18] APIs are still a work in progress, but most of them can be already tested by developers.

4. IDENTITY

Mozilla introduced its sign-in system for the Web. Mozilla Persona [21] (previously known as browserID) is an alternative to Google Login or Facebook Connect. It is the easy way to sign-in by using any of user's existing email addresses. It is a big advantage because email is already a fully-distributed system and probably every person surfing the Internet has an email address. Email address can be also verified independently, without the participation of an email provider.

Persona makes adding authentication features much easier for the developers, since implementing it requires very little code and there are some libraries already provided. It also has multiple advantages for users, like more control over information they share. The user experience is supposed to be streamlined one-click with the least amount of confirmation messages. There is no need of site-specific passwords or registration. It also allows users to manage their different e-mail addresses and to keep their identities separate (for example possibility of having different identity for professional and private usage). The whole authentication is done only by e-mail address, identity provider does not need any additional information (like phone number for OpenID) and it is not aware of user's identity. Mozilla Persona implements BrowserID Protocol. It also supports Gmail and Yahoo addresses by using the identity bridging.

4.1 SIGN IN PROCESS

The sign in process depends on what email provider a given user has. If a user has a Gmail or Yahoo account, Persona uses the identity bridging. Instead of using a mechanism such as sending confirmation emails, Persona verifies user's

identity by using email provider's existing OpenID or OAuth gateway [35]. As a result, the user has no additional action to do. He just needs to confirm that he wants to use his email account for Persona. Of course if the email session is closed he needs to login.

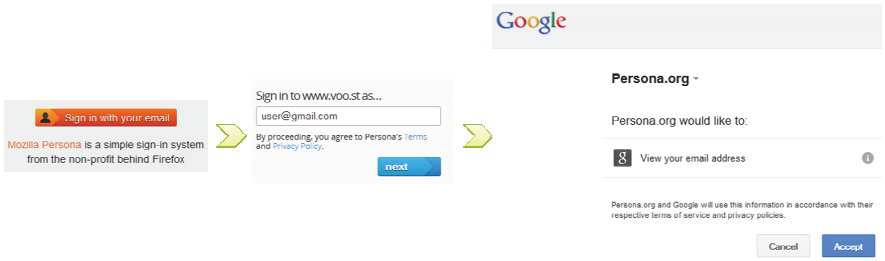

Figure 6: Persona sign in - identity bridging

If the user uses any email provider different than directly supported one, email verification is necessary. So whenever the user wants to add a new email address, a verification email is sent to it, and the user needs to click on a link in the received message.

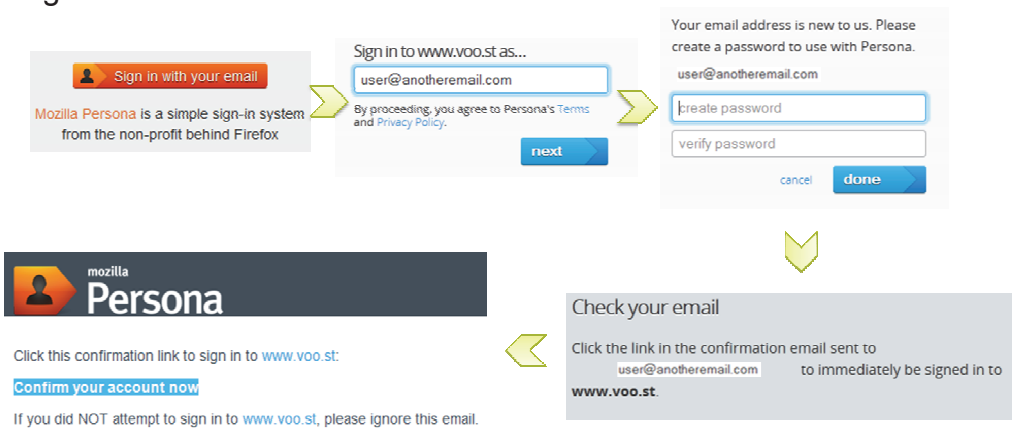

Figure 7: Persona sign in - any email provider

After first verification, email addresses are accessible whenever the user logs into Persona by using any of them. When a user wants to log into a given service he can pick one of his email addresses.

4.2 BrowserID Protocol

BrowserID [22] is a universal login system. It does not require email providers to support it, so the whole system is more flexible and open. It uses an identity authority that maintains a list of email addresses previously verified – it proves user's control of an email address to a given website. The identity is verified locally, without giving any information to third parties. The identity provider cannot track user's activity thanks to a certification system.

Persona implements BrowserID protocol to authenticate a user. Email addresses are used as identities. Any email may be used even if it is not managed by a trusted site that proves users' identity for a given website. Protocol overview website [23] describes three principal actors:

- Users that want to sign in using persona,
- Relying Parties (RPs) – websites that allow users signing in by using Persona,

- Identity Providers (IdPs) – domains that can assure users' identity by issuing Persona-compatible certificates.

There are three essential steps in the sign in process:

- User certificate provisioning,
- Assertion Generation,
- Assertion Verification.

4.2.1 Sign in flow

The whole flow is presented on the figure below.

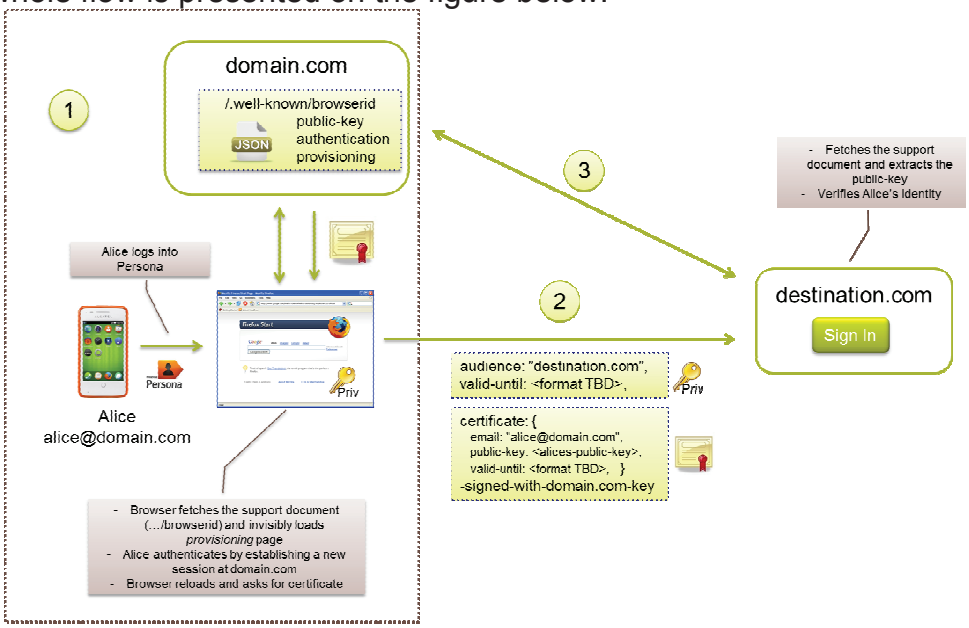

Figure 8: BrowserID Protocol - sign in flow

User certificate provisioning (1)

In order to sign into a website, a user needs to prove that he is the owner of the email address that he uses to sign in. As a result a cryptographically signed certificate is created by an IdP. Standard public key cryptography is used.

User's certificate contains: email address, user's public key for a given email address, issuing and expiration time of the certificate, IdP's domain name. The certificate is signed by the IdPs private key.

Since any user can use several email addressed within his Persona account, there is a different keypair generated for each of them. The certificate needs to be regenerated whenever it expires or whenever the user changes browser or computer.

To obtain a new certificate, the browser goes to /.well-known/browserid and gets a support document from the identity's domain. It fetches it over SSL in order to assure the security. This document is a JSON file [24]. If a domain acts also as IdP that file contains: domains public key, authentication URL (page allowing users to log on) and provisioning URL (page for certifying user's identity). If it delegates the authentication to another domain the file should additionally contain an authority entry.

```
{
"public-key": {
"algorithm": "RS",
"n":"828189054051051344101872274958853916092212880155660785421174093731921063829933
065372736775574820852047369750675671118310059213229911271650133404435637133859834 5
631188680121124149247071157632213057727857552920284005275361257606145056058810213 9
907846854501252327551303482213505265853706269864950437458242988327",
"e": "65537"
},
"authentication": "/browserid/sign_in.html",
"provisioning": "/browserid/provision.html"
}
```


Figure 9: Support document

The browser uses the provisioning URL and loads it in an invisible IFRAME. By doing that, it delivers to this URL any cookies previously set and it makes available for JavaScript code any local storage corresponding to the given origin. Thanks to this data, the script that determines if a user is properly authenticated.

- If the user is properly authenticated – browser generates a keypair for the email used for signing in.
- If the user is not authenticated – the authentication URL is loaded and user can authenticate within the domain. When the authentication is finished successfully, the browser generates a keypair for the email used for signing in.

Later the browser passes the user's email address and its public key to the IdP in order to receive a signed certificate. The IdP creates and signs a user certificate and sends it to the user's browser.

Assertion generation (2)

The user certificate provisioning step allowed proving the link between an email address and a public key. The next step is to prove that the user is the owner of a corresponding private key. An identity assertion is created. It contains the origin of the relying party that the user wants to sign in to and its expiration time (usually several minutes). This document is signed by the browser with the private key corresponding to the email used for signing in. The user certificate and signed assertion are sent to the relying party for verification.

Assertion verification (3)

User certificate and identity assertion are used to provide user's identity. If either of them is wrong, the assertion is rejected. Secondly, the identity assertion's signature is verified by using the public key inside the certificate. If the verification is successful, the relying party knows that the current user is the owner of the key sent in the certificate.

The relying party also needs to check that the certificate is valid. It gets identity provider's public key from /.well-known/browserid document. With this key it verifies the signature on the user certificate.

As a result the relying party is assured that the certificate is legitimate and that it matches the user that wants to login. Also whenever a user asks an identity provider about the certificate, it does not precise a website that it wants to use it for, so user's actions are not tracked by the identity provider.

4.2.2 Mozilla Persona - work in progress

Mozilla Persona is still under development. Its biggest advantage is the possibility of using any email address. It is different from existing systems, like Facebook Connect or Google Login because a user does not need to create a login

especially for Persona. At this point the biggest issue that Mozilla is facing is the lack of websites using their system. Although thanks to existing libraries it is relatively easy to add Persona support to any website so this system may be easily adapted by developers. Persona is also supported in Firefox OS phones, so it is logical to include it in web applications.

5. PAYMENT

Mozilla with its Firefox OS is trying to redefine the way the payments work for mobile applications. They want to change the way the stakeholders cooperate. In Mozilla's offer there are no restrictions imposed by business models or by central controlling authorities.

Note: The research presented in this chapter is valid for Firefox OS v.1.0. In September 2013 after several tests navigator.mozPay() API was proposed to be deprecated and to be replaced with lower-level primitives that can be directly used by the providers [31].

Users and developers are not locked into one marketplace belonging to Firefox OS. Mozilla gives the possibility of implementing different marketplaces and payment systems. Work on payments is still in progress, although Mozilla is already working with mobile phone operators to support payment with mobile phone bills. It also wants to address the needs of creditless users (users that don't have credit cards) and allows working with different types of payment providers in order to accept micro-payments.

This strategy is in line with the choice of Mozilla's target users. Since Firefox OS targets users that don't have Smartphones yet, it wants to offer them easy, clear payment methods. For example in Poland (the second country in which Firefox OS phones were launched) people are not so keen on providing their credit card information, most of them have only debit cards that do not always allow online payments. In the diagram below it is shown that for online shopping clients mostly use bank transfers. Credit card payments represent less than 7% of Internet payment transactions. That is why offering payments with mobile phone bills and addressing creditless users is an advantage of Mozilla's strategy.

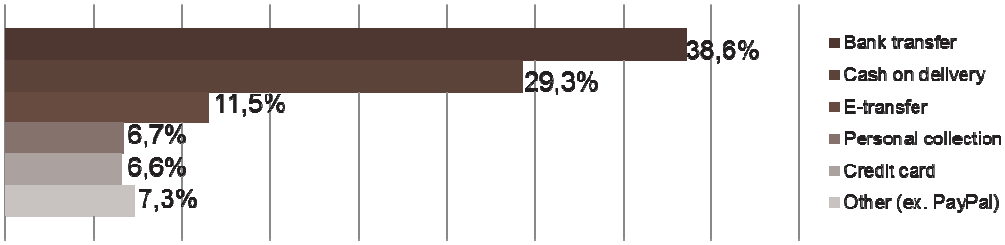

Figure 10: Online Payments in Poland in 2012

Mozilla wants to reorganize the payment process and make it more flexible for users but also for businesses [27]. Nowadays users cannot choose how to pay, they are limited to predefined payment options. Usually they need to type in their credit card information on any website they use for online shopping – that is not the safest solution. On the business side it is not ideal either – merchants need to set up their payment processor that usually is costly, time consuming and may be complicated for not advanced developers.

Mozilla decided to introduce their mozpay() API, its aim is to make payments easy and secure. It is similar to google.payments.inapp.buy [27] but it adds the possibility of having multiple payment providers and carrier billing.

There are two types of payments introduced by Firefox OS. There is a possibility of offering paid apps or in-app payments. In both types whenever Mozilla is participating in the payment process, it takes a 30% of revenue share. The 30% revenue is divided between Mozilla and its partners (payment processor – Bango and eventually a mobile carrier). The rest of the revenue belongs to the developer.

Mozilla uses Bango [28] as a payment processor. Bango assures payment experience across different platforms and expands one-click payment reach. Thanks to it Firefox OS may offer different payment options like operator billing or credit card billing. It provides a billable identity that allows unique user identification, including carrier information in case of operator billing. Since Firefox OS exists in many countries, Bango takes care of operator and country regulations.

Bango provided three APIs customized for Mozilla [29]:

- Mozilla Exporter API – allows registering developers, so they can sell apps or access in-app payments;
- Billing Configuration API – creates an one-time token that can initiate a payment;
- Mozilla Vendor Portal API – allows registered developers to access information about their payments.

5.1 PAID APPS

Mozilla implemented a system that allows creating paid apps with a receipt protocol. The receipt allows selling web apps that will run on any web device – "buy once, run everywhere" principle.

Web Application Receipt is a cryptographically verifiable proof of purchase – a digitally signed JSON file that can be read by clients and servers.

"typ": purchase-receipt, developer-receipt, reviewer-receipt, test-receipt
"product": JSON object identifying the product
"user": JSON object that store can used to identify the user with this receipt
"iss": full domain of the store that issued the receipt
"nbf": "not-before" timestamp, indicates when the purchase was completed
"iat": "issued-at" timestamp, indicates when this receipt was issued
"exp": "expiry" timestamp, indicates when this receipt will expire
"detail" (optional): URL to additional info about the purchase
"verify" (optional): URL that can be used by an authenticated application to verify a "receipt", allows also confirm the status of the payment with the issuer
"reissue" (optional): URL to re-issue a new receipt

Figure 11: Web Application receipt

In order to allow running an application on a given device a receipt validation is necessary. The receipt is normally sent to user's machine after a completed transaction. It can be validated either when the users starts the apps or on a regular schedule. To facilitate the process, Firefox offers a validation service. Also for developers' protection, all authorized stores are "whitelisted" in the app manifest so the app can't be sold in an unauthorized source.

Receipts are portable so the user can copy them between devices himself. To automatize the process Mozilla develops a service that allows syncing. Apps-In-The-Cloud API is a web service used to store and retrieve structured JSON records for keeping track of user's apps and devices.

So far there are no paid apps on the Firefox Marketplace but certainly it will change in the future.

5.2 IN-APP PAYMENTS

Mozilla strongly recommends app previews either by using in-app payments for paid features or by providing free and premium versions. Recurring subscriptions are planned for the future.

Mozilla proposed a Web API called WebPayment API. Its role is to initiate a payment for digital goods and receive a confirmation from a payment provider. The figure below shows three important parts in the payment process.

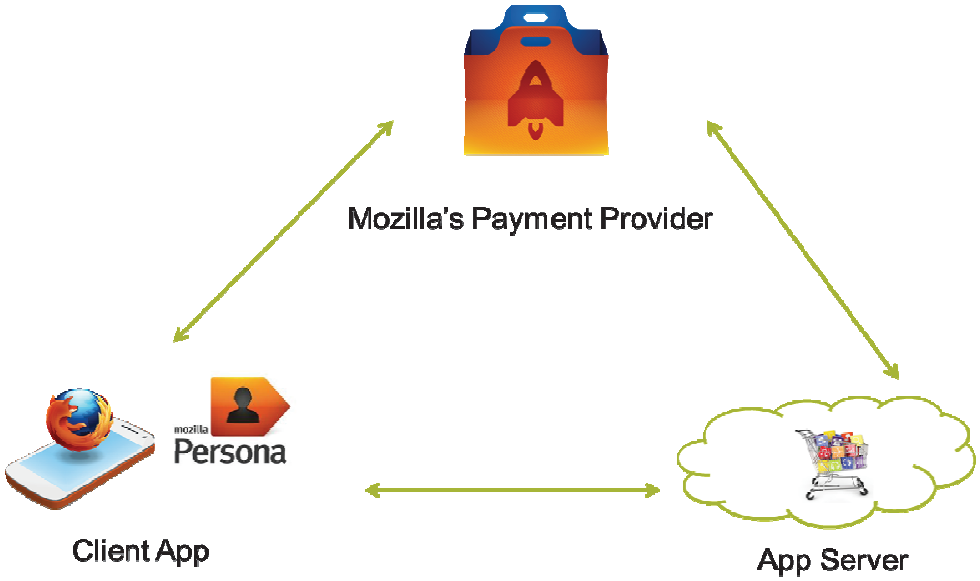

Figure 12: Payment Process – who is who

5.2.1 Payment Provider

Payment Provider implements WebPaymentProvider API. It verifies incoming information and signatures. Its main role is to process a payment from customers and to send funds and server notifications to the merchant by also providing a payment confirmation.

So far there is only one payment provider – Mozilla's Payment Provider. As it was discussed before Mozilla uses Bango as a payment platform. There are also some on-going implementations of WebPayment API like for example Web Pay [30].

5.2.2 App Server

App server is a server that belongs to a developer and it contains all application logic. In order to set up in-app payments it needs to register and exchange the information with the payment provider, for now with Firefox Marketplace.

App server communicates with the Payment Provider in order to set up payment details. It also generates a payment token that allows launching the in-app payment and gives all necessary payment information. It sends the purchased goods to a user whenever the payment process is finished.

To use in-app payments a developer needs to log into Firefox Marketplace Developer Hub. This will allow him to set up payment details (prices, bank account information, etc.). Later he generates an Application Key (a public identifier that allows a Payment Provider to identify the application) and an Application Secret (shared between a Developer and a Payment Provider – must be securely protected).

5.2.3 Client App

Client app allows users to buy different goods. The app gives user the information about possible goods with corresponding prices. Whenever user decides to make a payment the mozpay API is called.

5.2.4 Payment Token

The most important element in a payment process is a payment token. It contains all the essential information concerning a good being purchased. It is an encoded JSON object, digitally signed using JSON Web Signature. It is send between all three parties throughout the payment process.

```
"iss": APPLICATION_KEY,   - application key, obtained from Firefox Marketplace
"aud": "marketplace.firefox.com",   - by default, never change
"typ": "mozilla/payments/pay/v1",
"iat": 1337357297,
"exp": 1337360897,
"request": {     "id": "915c07fc-87df-46e5-9513-45cb6e504e39",    - should be unique for each of app's product
"pricePoint": 1,     - pricing table available on Mozilla's website*
"name": "Magical Unicorn",
"description": "Adventure Game item",
"icons": {
      "64": "https://yourapp.com/img/icon-64.png", },
"productData": "user_id=1234&my_session_id=XYZ",  - used for identifying a transaction
"postbackURL": "https://yourapp.com/payments/postback", - used to send payment process information
"chargebackURL": "https://yourapp.com/payments/chargeback", - used to send payment process information
"defaultLocale": "en",   - language of the app
"locales": {    - other languages that app can be used in
      "de": { "name": "Magisches Einhorn",        "description": "Adventure Game Artikel"
```

Figure 13: Payment Token

5.2.5 Payment process

The payment process is presented on the figure below.

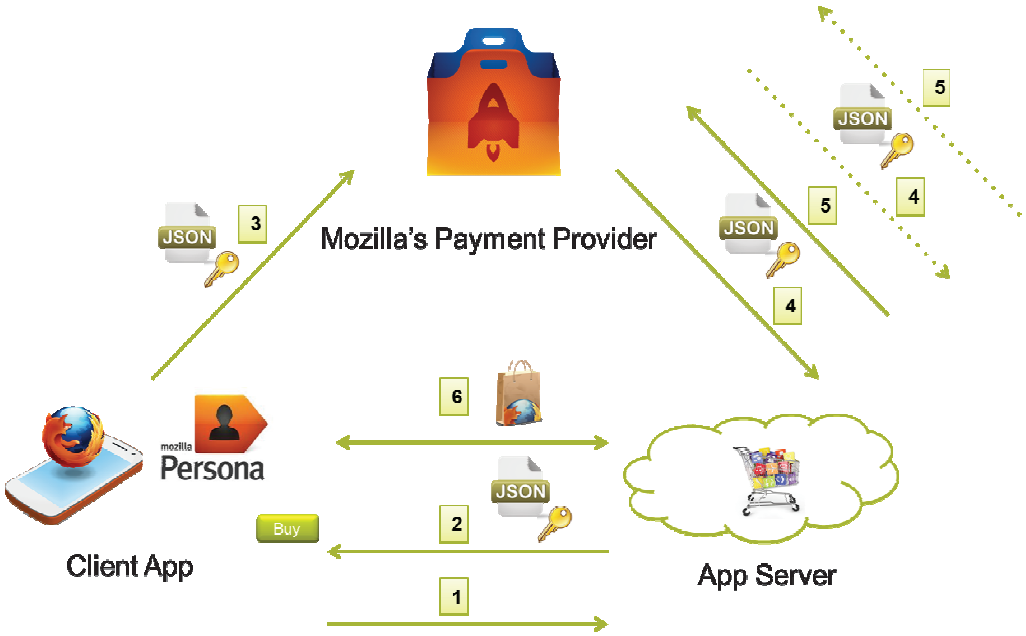

Figure 14: Payment Process

1. When a user decides to purchase something within a client app, he clicks a “Buy” button. It triggers a function that sends a request to an App Server and waits for a payment token to be signed.
2. Server generates the payment token and signs it with a private key so that information cannot be modified by unauthorized party. Later it sends the token to the user.
3. When the user receives the token it passes it to the payment provider by calling the mozpay API. When the navigator.mozPay() is called, the device shows a secure window that allows user to authenticate: login in to persona, enter the PIN and choose a payment method (credit card, payment billing, etc.).

The Payment Provider processes the payment. There can be two results of this operation:

4. Postback – meaning that the payment was completed successfully. It is a JSON file containing the original request with additionally a Mozilla specific transaction ID. It is sent to the postbackURL defined by the server in the payment token. To assure the security it is signed with the application secret.
5. Chargeback – meaning that the payment was not completed successfully. It is a JSON file containing the original request with additionally a Mozilla specific transaction ID and a reason why the payment was not completed (refund or reversal – buyer asked the credit card issuer to reverse a transaction). It is sent to the chargebackURL defined by the server in the payment token. To assure the security it is signed with the application secret. At this point chargeback is neither fully developed nor defined.
6. Whenever server receives the postback or a chargeback it acknowledges it by sending a HTTP Response to the Payment Provider. If the payment process was completed without an error it sends a purchased good to the user.

5.2.6 Work in progress

The payment model presented by Mozilla and the mozpay API are still in an experimental phase. In September 2013 after several tests navigator.mozPay() API was proposed to be deprecated and to be replaced with lower-level primitives that can be directly used by the providers [31].

The way mozpay functions is too limiting for the providers. It imposes a certain transaction flow that may not be convenient for all parties, since some of payment providers already have their own payment flow. Also mozpay uses a whitelist of providers, so whenever a new provider will like to offer its services, it would have to be approved (by for example Mozilla).

The new solution is to securely expose mozPaymentProvider primitives that websites can use to implement mobile payments. The subject is still under discussion.

6. CONCLUSION

Firefox OS is still in the early phase. Different features and functionalities keep changing. The existing applications are not all working smoothly yet but the marketplace keeps growing.

The strength of Firefox OS is that it is free from any proprietary technology, flexible and open. The web applications are said to have lower performance than native apps. On the other hand, using only web technologies offers the simplicity that allows lower entry level for developers. There is no vendor control and no device fragmentation, so developers are not forced to develop the same app for different devices – as long a WebAPIs proposed by Mozilla become standard. The aim of providing a “developer friendly” environment is to quickly grow the marketplace.

On the mobile market there is a strong presence of iOS, Android and Windows Phone. However Firefox OS is not targeting the same users, so it has a possibility to find a particular place on the market. There are new players like Ubuntu Touch or Tizen that can threaten Mozilla, but the advantage of Firefox OS is that it is already in use and gained a certain amount of users. Another threat is that Firefox OS phones maybe be considered as limited and low cost, not a high end player, but its thanks to attractive prices Mozilla will be able to enter emerging, dynamic markets.

In October 2013 Mozilla announced Firefox OS v.1.1 that adds different features like MMS, push notifications and other. This upgrade will allow Mozilla to collect information about how many users are currently using its platform [32]

Also Firefox OS phones are about to be sold in new countries [32]. Telefonica will offer Firefox OS phones in Brazil and three other Latin America markets in the last quarter of 2013. Deutsche Telekom started the sales in Germany with its low cost carrier – Congstar. They are also planning to launch Firefox OS phones in Greece and Hungary sometime soon. Last but not least, Telenor by the end of this year should launch the sale of Firefox OS Phones in Hungary, Serbia and Montenegro.

The opinions about Firefox OS are divided. Although it is difficult to compare it to iOS or Android, because those are operating systems, that have been on the market for much longer. The future of Firefox OS will depend on how fast it will improve and how quickly the new quality applications will be added to the marketplace.

7. BIBLIOGRAPHY

[1] Mozilla Developer Network: Firefox OS, https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS

[2] Tizen Official Website, https://www.tizen.org/

[3] The Orange Notebook: Testing Ubuntu Touch: The final month before release, http://www.theorangenotebook.com/2013/09/testing-ubuntu-touch-final-month-before.html

[4] Jolla Official Website, http://jolla.com/

[5] Negative Gravity: Firefox OS _ another Mobile OS, http://negative-gravity.blogspot.fr/2013/03/firefox-os-another-mobile-os.html

[6] Mozilla Developer Network: Gaia, https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS/Platform/Gaia?redirectlocale=en-US&redirectslug=Mozilla%2FFirefox_OS%2FGaia

[7] Mozilla Developer Network: Gonk, https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS/Platform/Gonk?redirectlocale=en-US&redirectslug=Mozilla%2FFirefox_OS%2FGonk

[8] Mozilla Developer Network: Gecko, https://developer.mozilla.org/en-US/docs/Mozilla/Gecko?redirectlocale=en-US&redirectslug=Gecko

[9] 28 days with Geeksphone and Firefox OS by Aras Balali Moghaddam, http://arasbm.com/geeksphone-and-firefox-os-1st/

[10] Mozilla Developer Network: App Center, https://developer.mozilla.org/en-US/docs/Web/Apps

[11] Mozilla Developer Network: WebAPI, https://developer.mozilla.org/en-US/docs/WebAPI

[12] Cult of Mac: Steve Jobs Was Originally Dead Set Against Third-Party Apps for the iPhone, http://www.cultofmac.com/125180/steve-jobs-was-originally-dead-set-against-third-party-apps-for-the-iphone/#cwD4X8eII039q6a7.99

[13] Wooga HTML5 Project Goes Open Source, http://www.wooga.com/press/releases/wooga-html5-project-goes-open-source/

[14] Entrepreneur: Mark Zuckerberg on Facebook's Triumphs and Tribulations, http://www.entrepreneur.com/blog/224415#ixzz2g5S3leEV

[15] W3C Interview: Financial Times experience with Web apps, http://www.w3.org/blog/2012/01/interview-financial-times-expe/

[16] Mozilla Hacks: HTML5 mythbusting, https://hacks.mozilla.org/2012/11/html5-mythbusting/

[17] Firefox Marketplace: Developer Hub – Application Validator, https://marketplace.firefox.com/developers/validator

[18] Mozilla Wiki: WebAPI, https://wiki.mozilla.org/WebAPI

[19] Mozilla Wiki: Simple Push, https://wiki.mozilla.org/WebAPI/SimplePush

[20] Mozilla Wiki: WebActivities, https://wiki.mozilla.org/WebAPI/WebActivities

[21] Mozilla Persona, http://www.mozilla.org/en-US/persona/

[22] GitHub: BrowserID, https://github.com/mozilla/id-specs/blob/prod/browserid/index.md

[23] Mozilla Developer Network: Protocol Overview, https://developer.mozilla.org/en-US/Persona/Protocol_Overview

[24] Mozilla Developer Network: .well-known-browserid, https://developer.mozilla.org/en-US/Persona/.well-known-browserid?redirectlocale=en-US&redirectslug=Persona%2F.well-known-browserid

[25] Chip.pl: Poradnik - jak bezpieczne płacić w Internecie, http://www.chip.pl/artykuly/porady/2013/08/bezpieczne-platnosci-internetowe-i-mobilne?b_start:int=0
[26] Mozilla Hacks: Introducing navigator.mozPay() For Web Payments, https://hacks.mozilla.org/2013/04/introducing-navigator-mozpay-for-web-payments/
[27] Google Developers: Wallet for digital goods, https://developers.google.com/commerce/wallet/digital/docs/index
[28] Bango Official Website, http://bango.com/
[29] Mozilla Wiki: Apps/ID and Payments: Bango API, https://wiki.mozilla.org/Apps/ID_and_Payments#Bango_API
[30] Web Pay, https://webpay.readthedocs.org/en/latest/
[31] Google Groups: mozilla.dev.webapi - proposal: replace navigator.mozPay() with primitives, https://groups.google.com/forum/#!msg/mozilla.dev.webapi/cyk8Nz4I-f4/5er6JojC3TsJ
[32] ZDNet: Tristan Nitot, Mozilla : "Firefox OS va progressivement monter en gamme", http://www.zdnet.fr/actualites/tristan-nitot-mozilla-firefox-os-va-progressivement-monter-en-gamme-39794749.htm
[33] PhoneGap: http://phonegap.com/about/
[34] FAQs about app manifests, https://developer.mozilla.org/en-US/Apps/Developing/About_app_manifests
[35] Identity at Mozilla, What is an Identity Bridging http://identity.mozilla.com/post/56526022621/what-is-an-identity-bridge

Campus de Brest
Technopôle Brest-Iroise
CS 83818
29238 Brest Cedex 3
France
+33 (0)2 29 00 11 11

Campus de Rennes
2, rue de la Chataigneraie
CS 17607
35576 Cesson Sévigné Cédex
France
+33 (0)2 99 12 70 00

Campus de Toulouse
10, avenue Edouard Belin
BP 44004
31028 Toulouse Cedex 04
France
+33 (0)5 61 33 83 65

www.telecom-bretagne.eu

Imprimé à Télécom Bretagne
Dépôt légal : novembre 2013
ISSN : 1255-2275

TELECOM
Bretagne