"Linked Open Apps Ecosystem to open up innovation in smart cities"
Project Number: 297363

Deliverable:	**D6.6 Pilots technical evaluation report final**
Version:	**1.3**
Delivery date:	**28th of September 2015**
Dissemination level:	**PU**
Author:	**UOC, CISCO**

Summary

This report, based on the work carried out in T3.2 (set of metrics to evaluate the platform in real deployment scenarios) will present the approach taken from the technical evaluation of the pilots.

DOCUMENT HISTORY

Version	Date of issue	Status	Content and changes	Modified by
0.1	1/7/2015	Draft	First draft	Frank Van Steenwinkel
0.2	3/7/2015	Draft	Revision	Ramon Ribera (UOC)
0.3	15/09/2015	Final	Final version	Ramon Ribera (UOC)

1. Introduction

This report, based on the work carried out in T3.2 (set of metrics to evaluate the platform in real deployment scenarios) will present the approach taken from the technical evaluation of the pilots. At the time of writing this document, we did not have enough applications developed, to test the platform under high load conditions as less than 10 apps are running on the platform.

The following areas have been evaluated:

- Security
    - XML-based Web services are becoming a more pervasive foundation technology for integrating applications and exchanging data in Service Oriented Architectures (SOAs). Like all new technologies, however, XML-based Web services also present new security challenges in the form of XML data structures, granular application calls, input data, or executable attachments, all of which can be maliciously constructed to damage or expose a receiving application. XML-based Web services compound the number of vulnerabilities by providing access to application APIs and target applications. The distributed, peer-to-peer nature of Web services also introduces bilateral threats and vulnerabilities that can be passed through multiple application hops. A complete threat-protection framework needs to address three key functions: Prevention, Protection, and Screening.
- Operations and management capabilities
    - Different user roles were evaluated on the iCity Platform. Several Administrative account levels have been evaluated.
- Reporting
    - Developer's user experience such as tracking API performance and reporting capabilities have been analyzed.
    - For Platform administrators, ranked and publisher reports have been evaluated.
- Performance testing metrics
    - Networking performance, iCity platform performance and performance characteristics of the connected infrastructure are mainly influencing the performance results.

TABLE OF CONTENTS

1. Introduction 3

2. Security Aspects 5

2.1 API Management and Security 5
2.2 Communication between the iCity Gateway and the back end server 8
2.3 Authenticating an API 8

3. Operational and Management Capabilities 9

3.1 Feedback from cities 9
3.2 API Portal 12
3.3 The Dashboard 15
3.4 Functionality by User Role 16
3.4.1 Administrative Roles (Internal Roles): 16
3.4.2 Developer Roles (External Roles) 17
3.5 Tasks Performed by User Role 18

4. Reporting 20

4.1 Accessing Reports 20
4.2 Developer Reports 20
4.3 Publisher Reports (Administrators) 21
4.4 Usage Reports (Administrators) 21
4.5 Ranked Reports (Administrators) 21

5. Performance Testing 23

5.1 Feedback from development community 23
5.2 Key Metrics 23
5.2.1 Requests per Second 23
5.2.2 Bytes per Second 23
5.2.3 Latency 24
5.2.4 Maximum Concurrency 24
5.3 Number of Users and Latency 25
5.3.1 Planning 25
5.3.2 The User Base 25
5.3.3 Application Design 25
5.3.4 Service Latency 26
5.3.5 Requests Per Second 26
5.3.6 Concurrency Calculations 27
5.3.7 Back End Processing 30
5.3.8 Performance Optimization 30

6. Summary 31

2. Security Aspects

2.1 API Management and Security

Publishing APIs online makes organizations subject to the growing threat of cyber-attacks. While network firewalls can provide some measure of protection from standard, Web-based attacks, they cannot address API threats because they lack the ability to deal with the messaging protocols used by APIs today, such as XML and JSON.

Managing APIs also presents a number of problems, primarily around creating, maintaining and updating different versions of APIs for different customers, as well as granting third parties the ability to aggregate and orchestrate across APIs to create new services and richer responses to queries. APIs are like any other piece of code that is created; they are developed, tested, deployed and revised as needed. However, moving Web APIs between environments or deploying new versions of APIs can expose hidden dependency issues or break your customers' existing integrations, causing downtime or even SLA violations. The iCity Platform is protected against attack and downtime.

The iCity Platform features a policy-based security model that is easily tailored to accommodate different security requirements for each API. It offers core policies that can be easily shared across sets of APIs to create a consistent basic security stance, as well as the ability to specialize on an API-by-API basis to meet the specific needs of a particular application.

APIs give client-side developers—both legitimate developers and potential system crackers much more finely-grained access into an application than a typical Web app. This is because the granularity boundary for calls to backend tiers moves from relatively secure internal tiers (those that reside safely in a DMZ) all the way out to the client application residing on the Internet. (See fig 1)

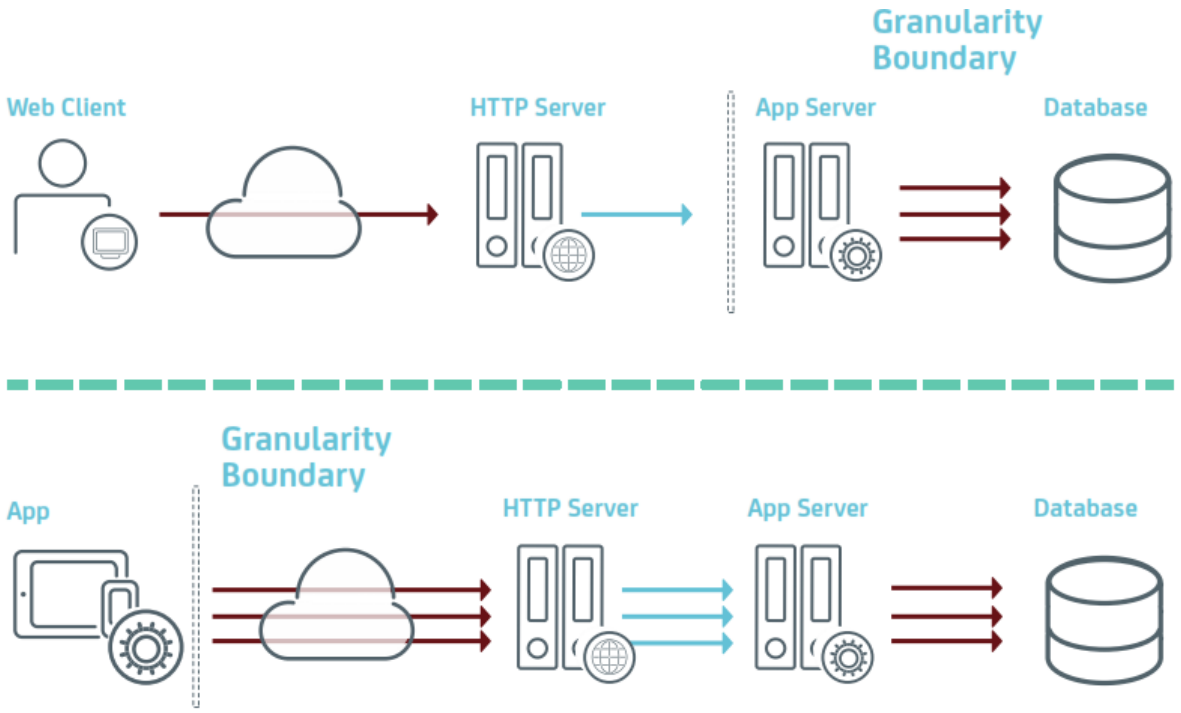


Figure 1 Granularity boundary

Most individual attacks against APIs fall into one of three broad categories:

1. Parameter attacks exploit the data sent into an API, including URL, query parameters, HTTP headers and/or post content.

2. Identity attacks exploit flaws in authentication, authorization and session tracking. In particular, many of these result in bad practices from the Web migrating into API development.

3. Man-in-the-middle attacks intercept legitimate transactions and exploit unsigned and/or unencrypted data. They can reveal confidential information (such as personal data), alter a transaction in progress or replay legitimate transactions.

By understanding these broad categories, we designed an effective mitigation strategy against API attacks.

In the iCity Platform, we should make SSL/TLS the rule for all APIs. Adding SSL/TLS—and applying this correctly—is an effective defense against the risk of man-in-the-middle attacks.

SSL/TLS provides confidentially and integrity on all data exchanged between a client and a server, including important access tokens such as those used in OAuth. It optionally provides client-side authentication using certificates, which is important in many high-assurance environments.

However, despite it being much simpler than message-level security methods such as WS-Security, SSL is still subject to misuse. It is easy to misconfigure on the server and research has shown that many developers do not properly validate certificates and trust relationships when applying it.

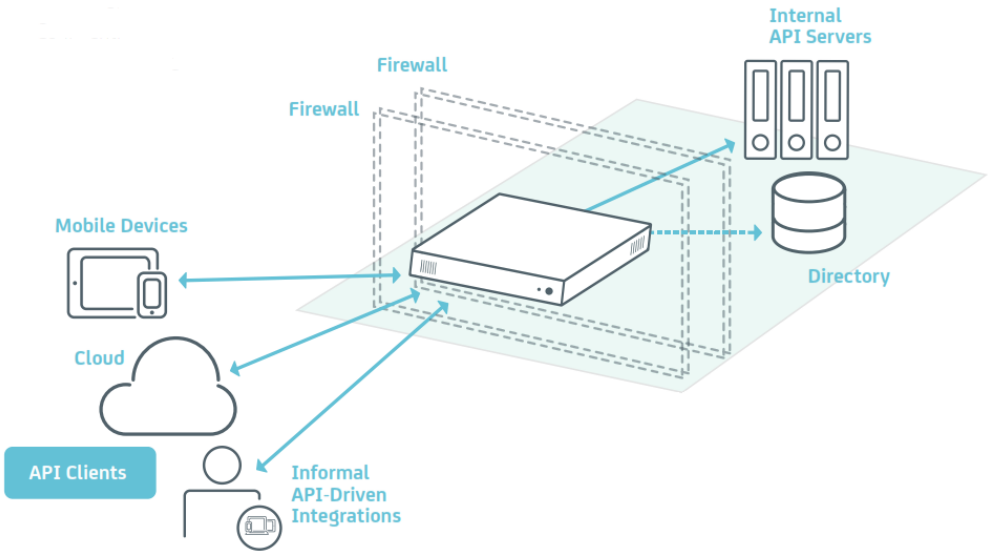


Figure 1 Server side API development versus API security

Our iCity Platform Gateway features a policy-based security model that is easily tailored to accommodate different security requirements for each API. It offers core policies that can be easily shared across sets of APIs to create a consistent basic security stance, as well as the ability to specialize on an API-by-API basis to meet the specific needs of a particular application. It also integrates easily with existing identity systems.

The iCity Platform gateway provides the security administrator with complete control over all aspects of API security, including:

• Threat detection, including SQL injection, XSS, CSRF, message size, etc.

• Confidentiality and integrity, including limitations on cipher suites, advanced ciphers based on technology such as ECC digital certificates, message-based security etc.

• Message validation, including XML and JSON schema validation

• Authentication support, including basic credentials, API keys, OAuth, SAML, OpenID Connect, Kerberos, x509 digital certificates, etc.

• Integration with most common identity and access management (IAM) systems

• Rate limiting and traffic shaping of transactions against any model

Although many application developers will leave security to the end of the project, rushing implementation, if they get to it at all. This behavior is difficult to change. Therefore, our iCity Platform is separating out API development and API security implementation which is the best practice to follow in order to keep it safe!

2.2 Communication between the iCity Gateway and the back end server

The communication between the Gateway and the back end server needs to be secured.

Authentication and Authorization of API access is done only by the Gateway (with developer management provided through the API portal). Protecting the API is the job of the Gateway, not of the back end server.

To avoid many back end system security reconfigurations, and to centralize the security rules, the Gateway (not individual developers) authenticates itself against the back end system.

Fixed 'username/password' basic authentication over HTTPS is probably the simplest method and might be acceptable at the initial stage. In other words we assigned the Gateway a username/password with which it authenticates itself against the back end server. Since basic authentication itself is not encrypted (only obfuscated through base64 encoding), it is strongly advised to use it over HTTPS. In that way at least the password is protected during transport.

2.3 Authenticating an API

The iCity Platform has a utility to enable developers to discover APIs interactively.

By making choices between APIs' valid resources and methods, and then submitting queries and viewing responses, developers can, amongst other things, gain a better understanding of how the APIs work.

In order for a request to be executed correctly, an API must be authenticated on the iCity Gateway.

Different authentication methods are supported in the iCity platform, however we only use API key today:

- API Key;HTTP Basic,OAuth 1.0,OAuth 2.0

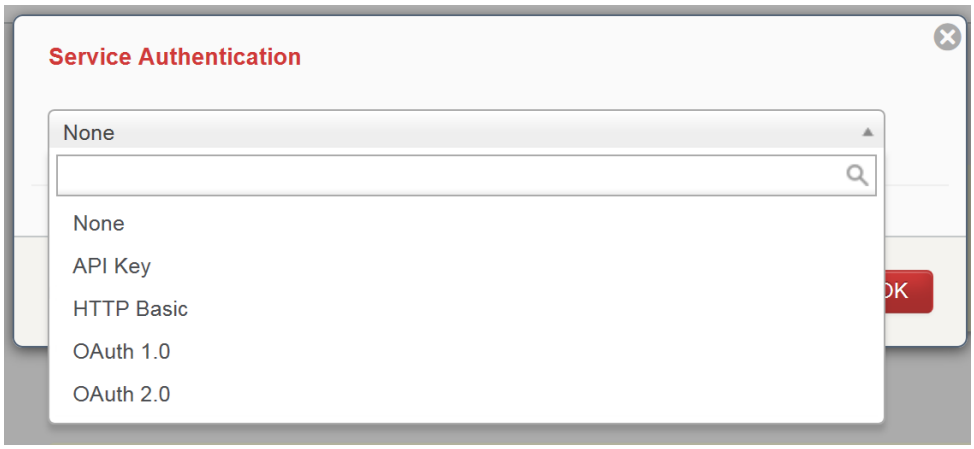


Figure 2 Service Authentication

3. Operational and Management Capabilities

3.1 Feedback from cities

As regards to potential stakeholders' perception about the purpose and the contents of the project, it is significant that *Co-creation* or *App development* are not chosen by participants in the iCamp activities as the main term or label, whereas they prefer other more vague concepts as *Smart City* or some kinds of *Openness*. This has also been detected in other similar projects: in the context of the Quadruple Helix model, participants are often not very acquainted with the essence of a co-creation process and with the role of users and the procedures to involve them. It is obviously not straightforward how to fit easily users' concerns and expectations to economic interests of developers. Probably, this could also be the case of the iCity project. Though the project is aimed at providing a tool for innovation in the development of apps for a better provision of services of public interest, by means of a cooperative model, most of the participants have not perceived this as the fundamental nature of iCity. Their perception seems to be clearly biased towards a more technological interpretation of the project.

We could also state that participants are much interested about open data and show low knowledge about open infrastructures –when asked about that, they show concerns about the feasibility of open platforms. Many are really interested in having open data integrated into the project's platform.

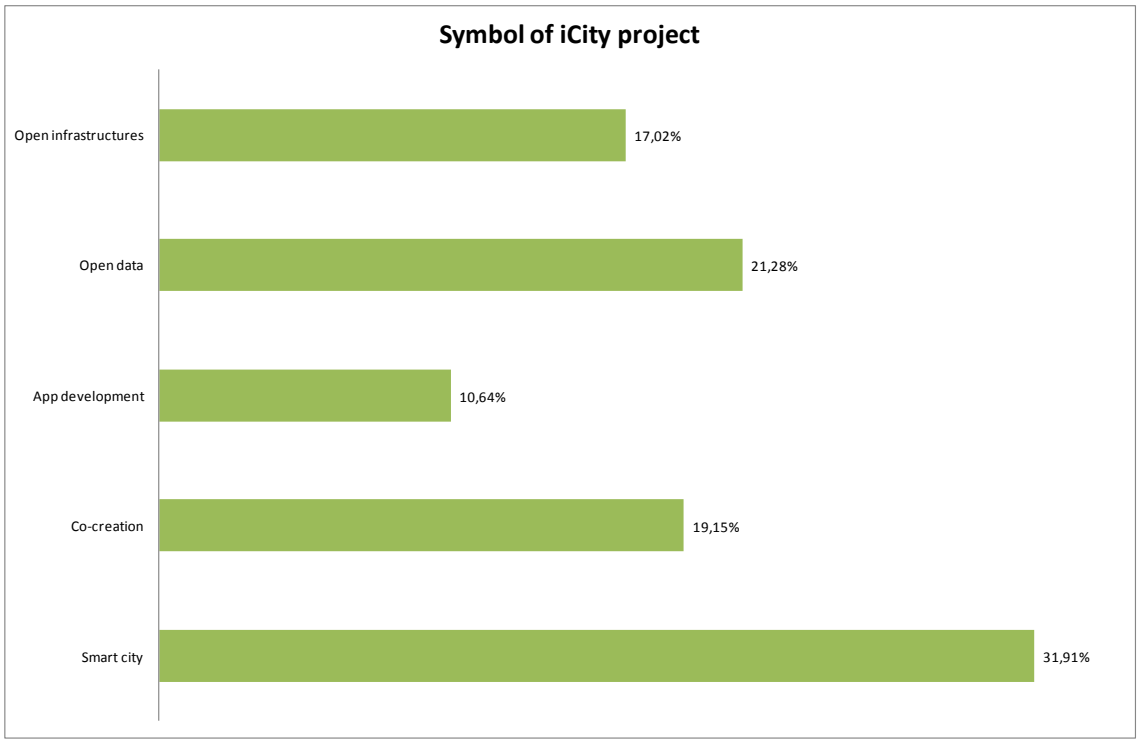


Table 1 Symbol of iCity

Significantly, the identification of iCity project with an innovation and app development process based on co-creation is lower among app developers. In fact, co-creation is not even mentioned by the people interviewed when asked about the project aims or innovative features. It can be stated, then, that the final aim of the project seems to be more comprehensible and obvious for researchers than for citizens or entrepreneurs. This is one of the main methodological challenges for this kind of projects. There is not a unique *narrative of the iCity project* detected among potential participants.

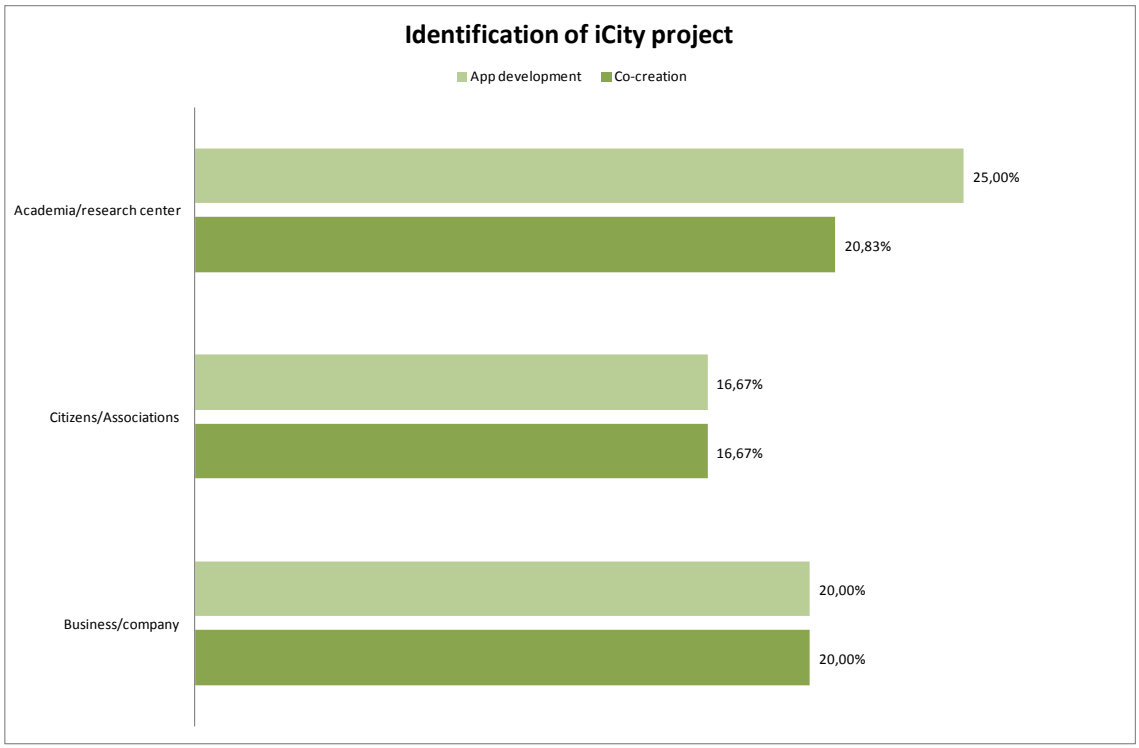


Table 2 Identification of iCity

Finally, it is also important to know if stakeholders are empowered and have an active role and involvement at this initial stage of the project. Generally, the engagement activities have been positively evaluated by potential stakeholders:

- Information provided was relevant (68.4%)
- Information met expectations and needs (84.7%)
- Training with the iCity API was useful (67.7%)
- iCity project team encouraged feedback (85.1%)
- Improvement of personal knowledge and skills (59.3%)

Despite this generally positive assessment, a few technical problems with the infrastructure in the meeting venues caused some frustration among participants. Moreover, some potential stakeholders were concerned about the usability and interactivity of the platform. The interface is often seen as too complex and most of them think potential users need to fill in too many forms. Some participants even say there is not enough information about the project management and that the work schedule is not realistic and defined enough.

Going beyond the engagement activities and moving forward with the project, most of participants clearly express their confidence in the platform. A large majority (76.9%) is convinced that iCity platform is an appropriate and helpful tool for designing and developing apps in the near future. Most of them (78.8%) think that interaction is a key factor to move forward.

About half of potential stakeholders (48.9%) in iteration meetings and 63.6% of people attending the iCamp activities state that engagement activities were useful for improving their professional networks and for getting to know the framework of the iCity project. The seeds for a new ecosystem for co-creation were actually put in the ground – something particularly apparent in the city of Barcelona.

Significantly, most of them (93.9%) confirmed their interest in attending forthcoming events related to the project. In fact, the overall rating of the project by potential stakeholders is very positive in the three cities and among all type of stakeholders.

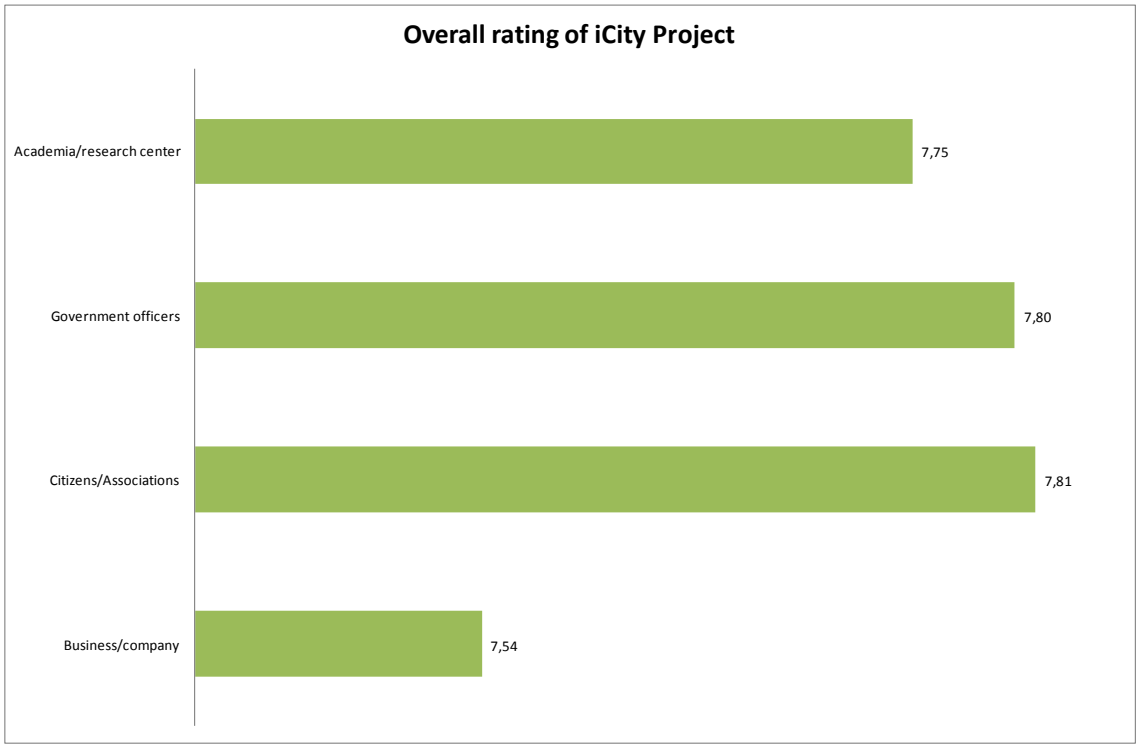


Table 3 Overall rating of iCity

Regarding the next stages of the project, several specific concerns and opinions have been raised from the interaction and interviews with potential stakeholders:

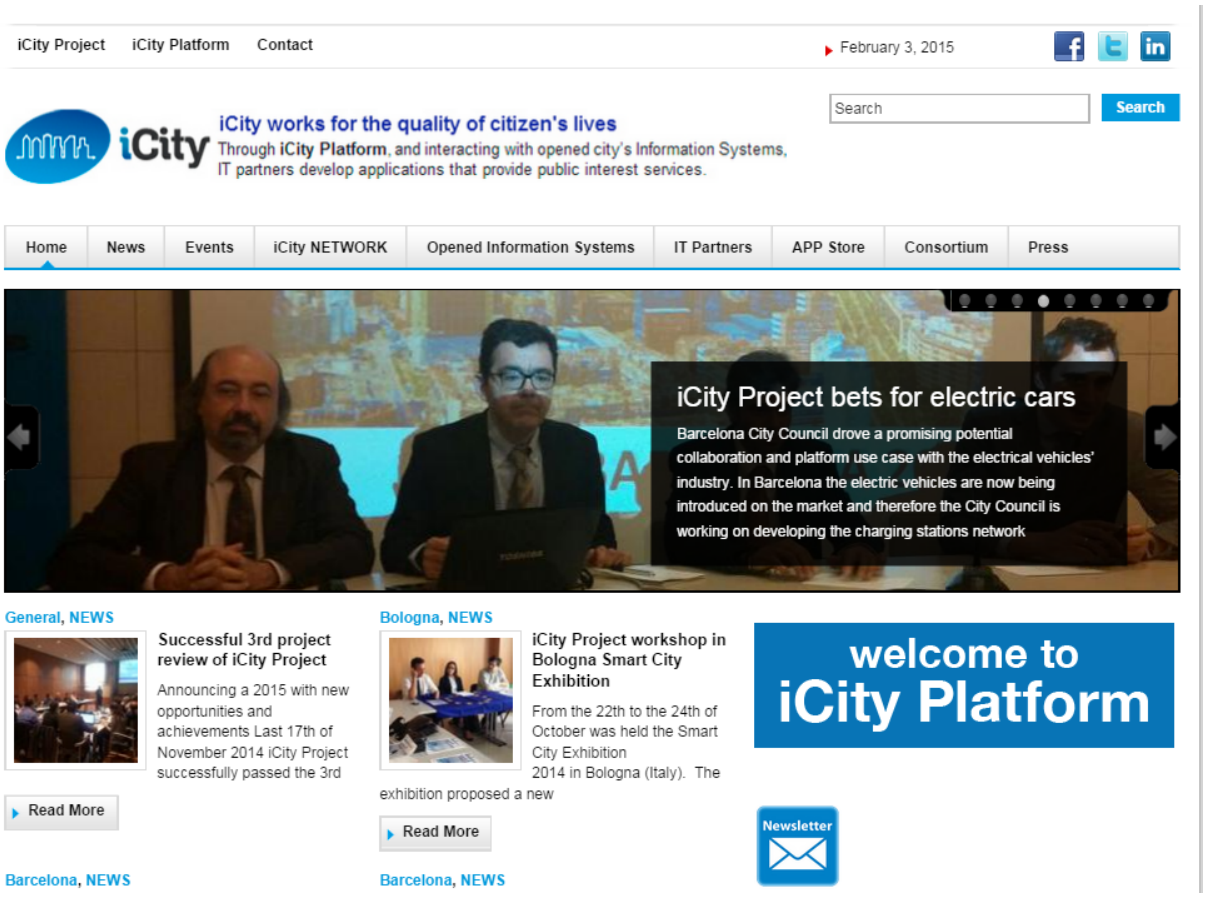
- Many participants – potential developers and thus people with a technical profile – asked very detailed questions about the platform. They were really interested in having dynamic and real time data available in the platform.
- Although they think the project is a great idea, developers are somewhat concerned about the available data in the platform. Some entrepreneurs, though showing some concerns about future revenues, see a great point in simplifying their demands on data

from public administrations. They also perceive very positively the fact that apps could be developed to be used in different cities. However, they are also concerned about the quality and homogenous format of data in the platform.

- People from public administration bodies would like to keep some kind of control on the data; they would like to know and track who and for what is using them. They see the project as a potential and very valuable tool for surpassing existing bureaucratic barriers between different city councils and sharing resources among them.
- Some independent participants believe there is a risk that apps will be basically centred on touristic and commercial services, not addressing other citizen concerns more linked to social and political issues.
- We have also found serious concerns about the legal issues involved in the project – mainly those related to the use of public data and the heterogeneity of procedures and norms among different public bodies. In particular, some of them are concerned about licensing issues. In fact, they ask for more normative information (policies, guidelines and standards) about the project.
- The fact that few services and little data were actually available is also linked to some kind of disappointment about the whole project. Showing some readymade apps would have helped – according to some participants.

3.2 API Portal

There is a public URL: icityproject.eu



If you have registered, depending on your login access rights, the iCity API Portal enables cities to upload APIs, manage, and report on third party developers and the applications they build using the APIs. (See Dashboard later)

The iCity Platform contains these two main functional areas:

- API Portal: This area is used to publish APIs, manage developers, and perform analytics/reporting.

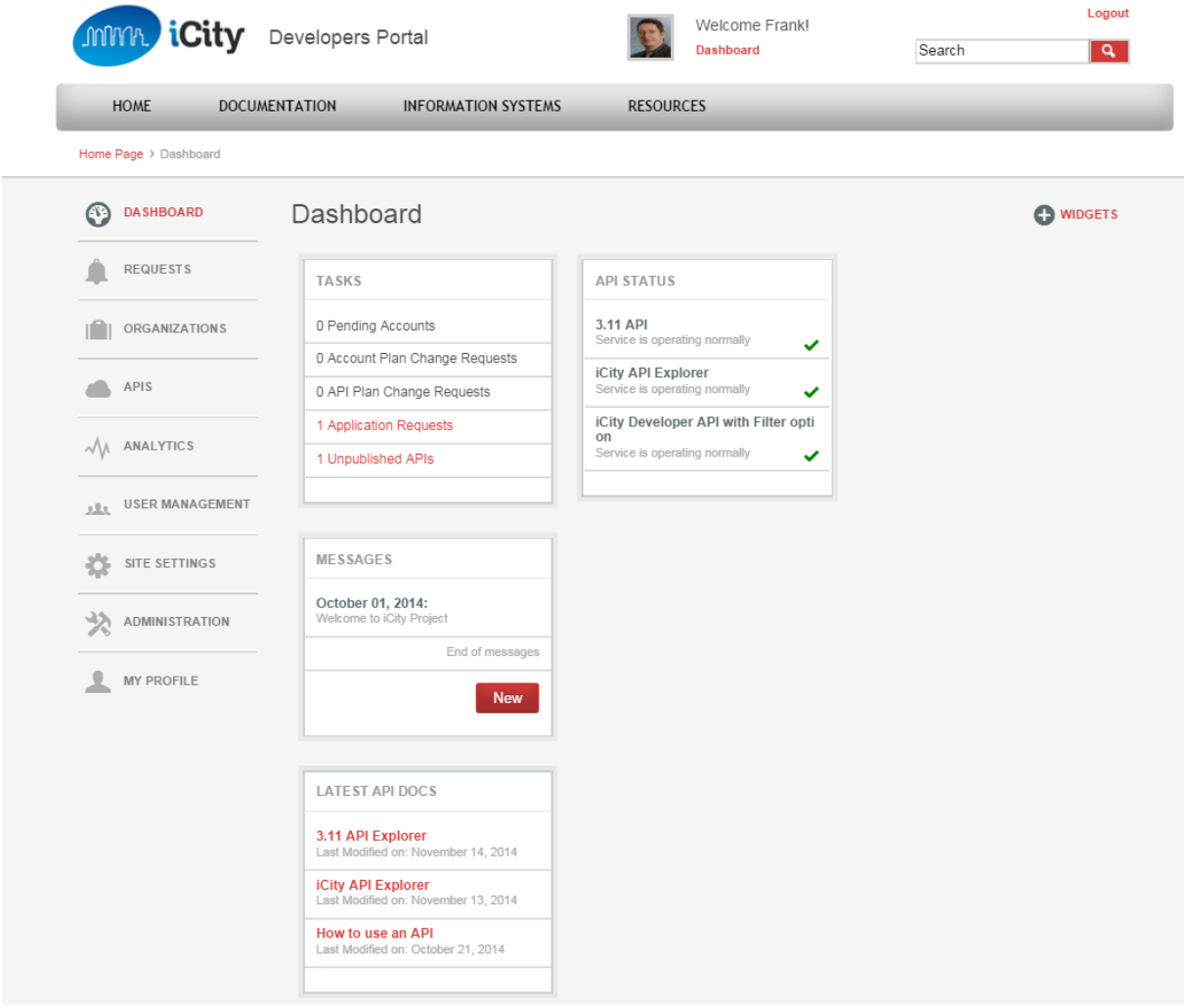


Figure 3 API Portal - Dashboard

- CMS (Content Management System): This area is used to set up accounts, work with CSS files, con system messages, and create Web content.

Administrator Account:

The iCity API Portal has a predefined Administrator account that can be used to log in and

create additional profiles via the content management system.

Changing a Password:

- Both developers and administrators are able to change passwords.
    - Users can change their own passwords on the dashboard of the iCity API Portal.

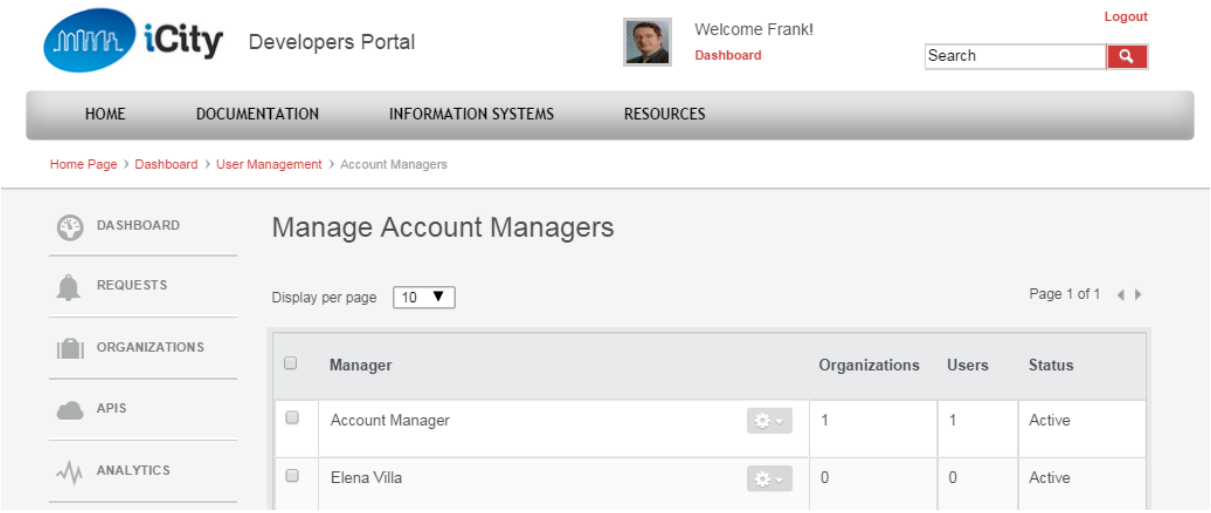


Figure 4 User management

    - Administrators can also use the CMS to change users' passwords, including their own.

      http://icity-devp.icityproject.com/admin is the URL for the CMS

- A maximum login attempt value is provided.

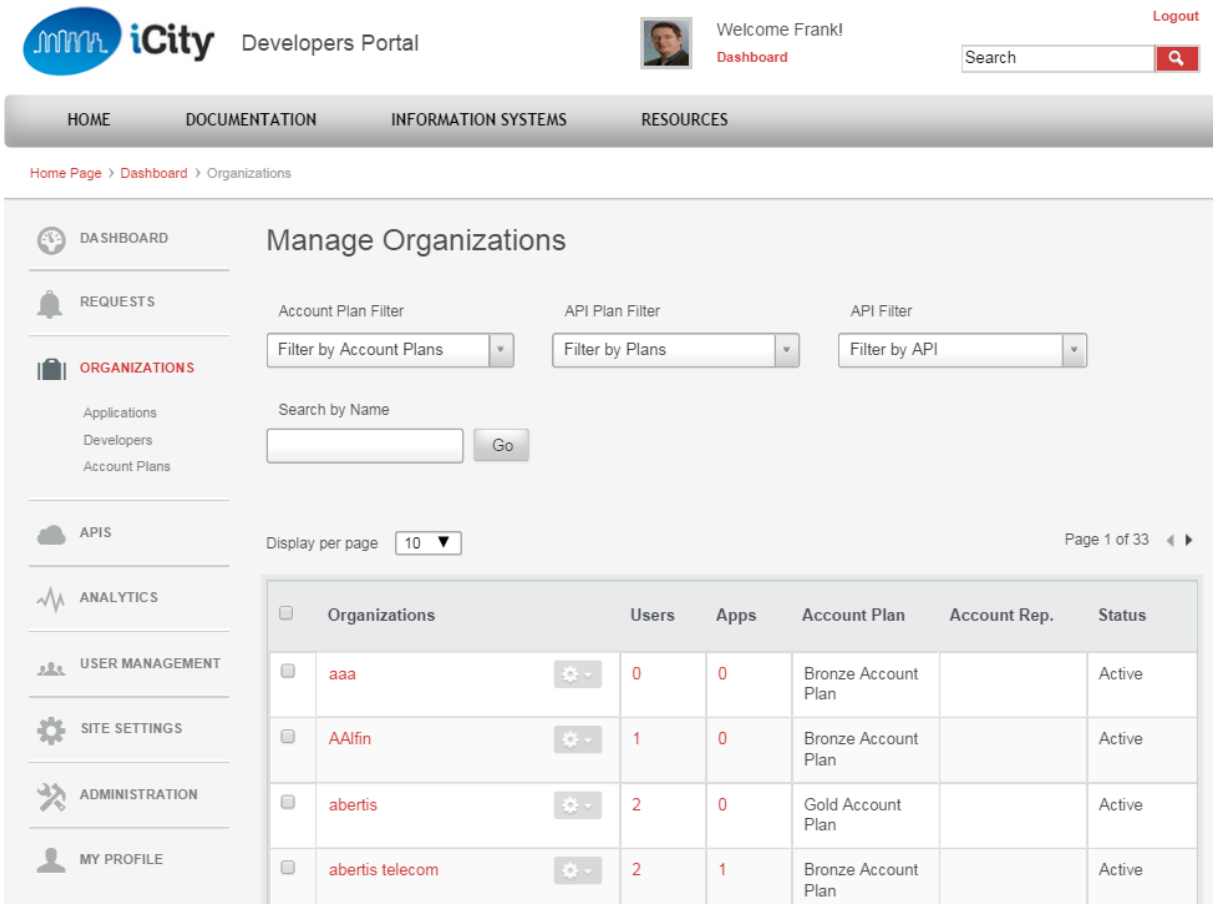


Figure 5 Organization Management

- The time accounts are locked and are configurable in minutes; hours, days.

3.3 The Dashboard

- The iCity Platform has a Dashboard; a primary interface for developers and several user roles pre-defined in the system: example: API Owners, Business Managers, and Account Managers.
- A navigation sidebar displays different links depending on the role of the logged in user. The Dashboard can be personalized by each individual user.
- Once you log in, the Dashboard is displayed by default.
- The iCity API Portal has the ability to create forums. Forums facilitate communication among all users in the API Portal. Forums should have different levels of access. Each level has permission to view different areas of the forum as well as perform different tasks.

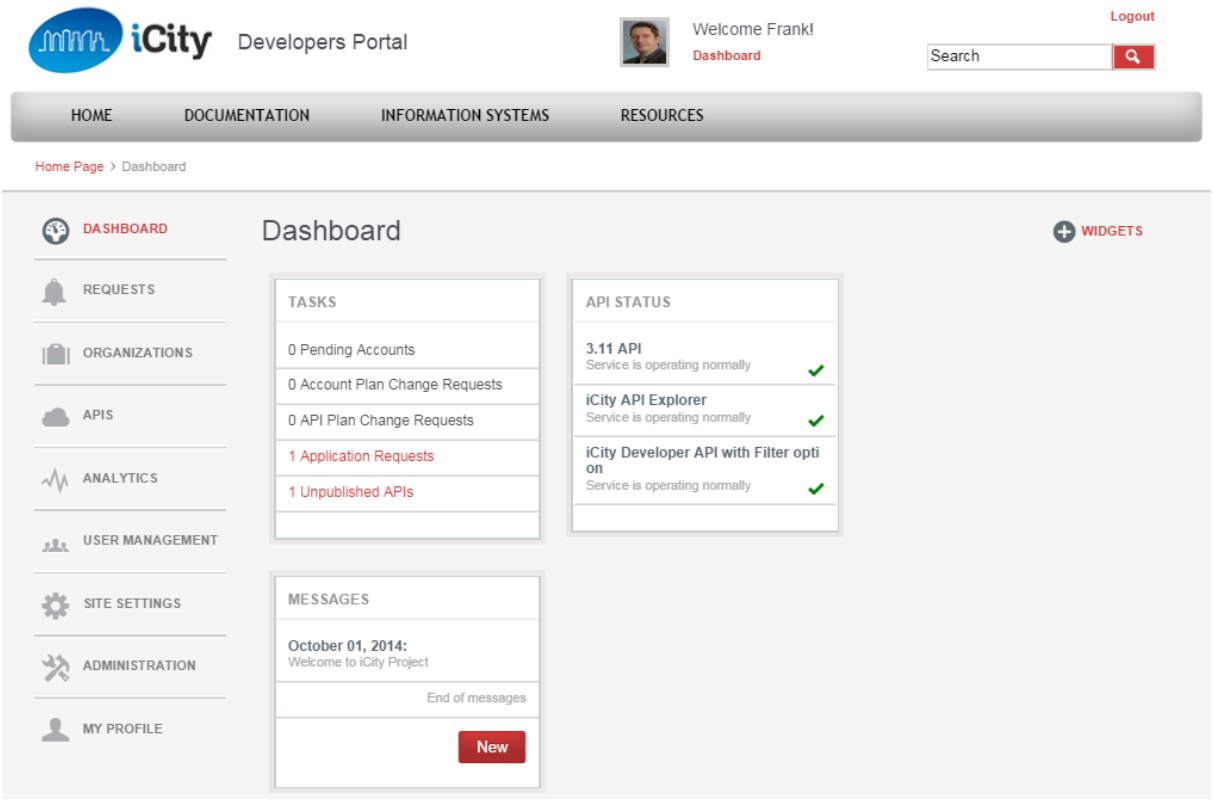


Figure 6 iCity Dashboard

3.4 Functionality by User Role

- The iCity API Portal is delivered with several user roles pre-defined in the system.
- These roles are defined as being either internal or external.
- Internal roles are created in the CMS and are internal and related to the business of implementing the portal, whereas external roles are accounts that must be invited by the system.
- Each user role sees different levels of navigation capabilities.

3.4.1 Administrative Roles (Internal Roles):

The following “internal” user roles are available in the iCity API Portal:

- **Administrator:**
  - The super user with access to all functionalities.
- **WebAdmin:**
  - The person responsible for setting up the API Portal, including:
    - Branding the API Portal
    - Creating and publishing the home page, documentation, and other

content

- **API Owner:**
    - The person tasked with defining, publishing and monetizing, or promoting the APIs. On the iCity API Portal, this person will be responsible for:
        - Defining API Plans (i.e., service levels) associated with each API
        - Publishing the APIs for use by developers
        - Measuring the effectiveness and usage of their APIs using the Analytics and Reporting feature

    The API Owner can also:
    - Manage organizations
    - Edit, enable, or disable applications
    - Email Organization Administrators

- **Business Manager:**
    - The person tasked with managing the developers who sign up to use the APIs. On the iCity API Portal, the Business Manager is responsible for the following tasks:
        - Defining Account Plans (i.e., technical support levels) that can be assigned to each developer
        - Assigning Account Managers to developers
        - Measuring the rate at which developers sign up
        - Ensuring that SLAs (Service Level Agreements) are being adhered to, by using the Analytics and Reporting feature

    The Business Manager can also:
    - Process requests (such as application requests, API Plans, and Account registrations)
    - Manage organizations (the same as API Owners)
    - Edit email templates and registration disclaimers

- **Account Manager:**
    - The person tasked with assisting the Business Manager with the developers. On the iCity API Portal, this person will be responsible for the following tasks:
        - Approving API and Account plan requests
        - Managing the developer's account on a daily basis
        - Managing organizations (similar to an API Owner)

3.4.2 Developer Roles (External Roles)

The following “external” user roles are available and preconfigured in the iCity API Portal:

- **OrgAdmin:**
    - OrgAdmin is the owner of an organization. Those are third-party users that sign up for an account in the iCity API Portal using the Registration Form. This person is responsible for managing his or her own organization and is usually the only developer or the first one to register for the organization.
- **Developer:**
    - A user that has been invited to join the iCity API Portal by an organization owner (OrgAdmin). These users are enrolled under the OrgAdmin's account. Developers are responsible for creating and managing new applications.

3.5 Tasks Performed by User Role

The following table summarizes the tasks each user is able to perform:

	API Owner	**Business Manager**	**Account Manager**	**Developer**	**Web Admin**	**Administrators**
View APIs	X					X
Publish APIs	X					X
Use or Designate Private APIs	X	X	X			X
Deprecate APIs	X	X	X			X
Add/Edit API EULAs	X					X
View and Message OrgAdmin	X					X
Create and Manage Account Plans		X				X
Request Account Plan Change				X		
Manage Account Managers		X				X
Manage Organizations (Access varies	X (for	X (all orgs)	X (only	X (if		X (all orgs)

by user role)	APIs)		assigned orgs)	OrgAdmin)		
Manage or Work with Applications (Access varies by user role)	X	X	X	X		X

4. Reporting

The reporting will be described in the future release of this document, once we have more applications running.

4.1 Accessing Reports

- Administrators are able to generate reports on APIs and organizations.
- Administrators are able to access ranked reports.
- Developers are able to generate reports on their applications.
- Developers with access to reports are able to view usage reports.

4.2 Developer Reports

Developers have access to both Application reports and API reports.

Usage and Latency reporting is available.

The Application reports allow developers to:

- **View the usage graph for an application**
- **View the latency for an application**

The API reports allow developers to:

- **View the usage for an API**
- **View the latency for an API**

4.3 Publisher Reports (Administrators)

Publishers should have access to both Application reports and API reports.

Usage and Latency is not only important for developers, but for publishers too as this will indicate somehow how end users will experience the application.

The Application reports should allow publishers to:

- **View the usage for an application**
- **View the latency for an application**

The API reports should allow publishers to:

- **View the usage for an API**
- **View the latency for an API**

4.4 Usage Reports (Administrators)

The usage report should provide a high level view of the Account Plan usage by organization.

4.5 Ranked Reports (Administrators)

A Ranked Reports page is available to all internal user roles that have access to Administrators, Business Managers, API Owners, and Account Managers.

This page provides a high level view of API usage or latency by application or organization.

- **Top Applications:**
  - View the applications that have the most hits against APIs. These are going to be the most popular applications.

- **Highest Latency:**
  - View the applications with the highest latency spikes over the time period chosen. The information here can help to perform troubleshooting.

- **Top Organizations:**
  - View the organizations that have the most hits against APIs. These are going to be the most valuable organizations.

- **Inactive Organizations:**
  - View those organizations that have generated no traffic or are no longer active on the Portal. These are accounts which might be purged.

	API Owner	**Business Manager**	**Account Manager**	**Developer**	**Web Admin**	**Administrators**
Manage or Work with Applications (Access depends	X	X	X	X		X

on user type)						
Approve/reject New Accounts		X				X
Approve/reject API Plan Requests		X	X			X
Assign Private API Access (to Developers)		X				X
Register for an account				X		
Add new applications		X	X	X		X
Access the Site Settings		X (email Templates and Registration only)			X	X
Access the Content Management System (CMS)	X				X	X

5. Performance Testing

For the performance testing, a lot of live tests with sufficient applications running (API calls) will be required. At the moment of writing this deliverable, we did not have enough applications running to test the platform in high load conditions. However, no complaints from development communities have been received related to the API call responsiveness of the platform. Complaints were related to the lack of available data (infrastructures) in the platform.

5.1 Feedback from development community

No complaints from development communities have been received related to the API call responsiveness of the platform. Complaints were related to the lack of available data (infrastructures) in the platform.

5.2 Key Metrics

5.2.1 Requests per Second

In terms of overall statistics, reporting systems often cover the number of requests served in a given month, or the worst case burst period request traffic. This implies metrics based on requests per unit of time and usually leads to throughput being regarded in terms of the number of requests per second.

Often requests per second are limited by networking issues:

- Network latency for very small messages can be a significant part of the whole request time overhead. This limits the number of new requests that can be accepted by the application stack in a given time period.
- Network bandwidth for large messages can limit requests per second as no more traffic can be put on the network interface. Modern hardware can easily swamp a Gigabit network with large messages.
- Some operations are time consuming and necessarily synchronous, like certain kinds of Lightweight Directory Access Protocol (LDAP) lookups, database queries, etc. Often, the only way to optimize this is to redesign the workflow to cache, or reduce these time wait cycles.

5.2.2 Bytes per Second

Some proposed benchmarks are based on measuring throughput in terms of bytes per second. For a variety of reasons these prove to be difficult to plan well. To accurately model a proposed application it is necessary to have knowledge of:

- Average incoming Message size
- Average back end Response time
- Maximum concurrency of back end systems

- Bottlenecks at Authorization and Authentication systems

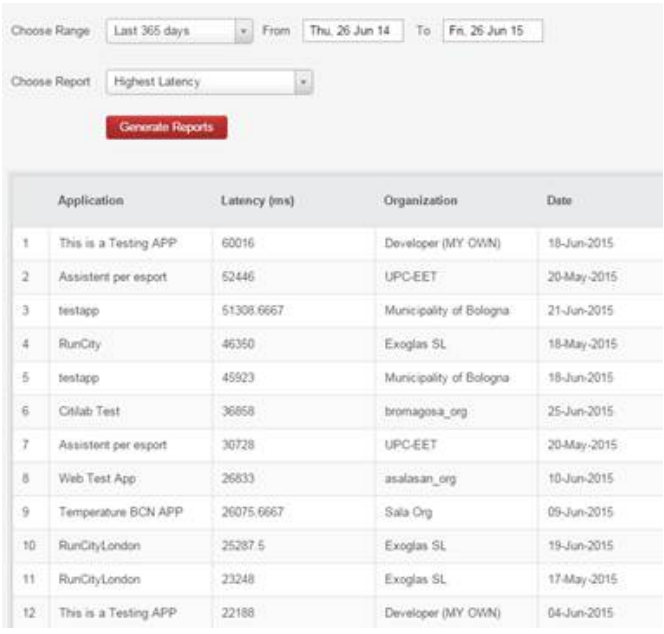
It is expected that the cities backend will be the bottleneck, it will be analyzed once we have more applications running.

5.2.3 Latency

Measuring the total elapsed time it takes a request to be serviced is critical in certain types of applications.

This has its own list of criteria that need to be taken into account:

- Usability of user interfaces is often enhanced with a faster response times.
- Latency becomes a performance benchmark especially in chatty applications that use a large number of requests to service a single user action.
- Technical people are often tasked with measuring this and the iCity Platform dashboard User Interface (UI) should feature instrumentation to show the separate components of request latency.
- It is important to note that latency and concurrency are often in opposition when building test cases.
- External decision points like LDAP, Single Sign On (SSO) systems often contribute latency to a whole application.



	Application	Latency (ms)	Organization	Date
1	This is a Testing APP	60016	Developer (MY OWN)	18-Jun-2015
2	Assistent per esport	52446	UPC-EET	20-May-2015
3	testapp	51308.6667	Municipality of Bologna	21-Jun-2015
4	RunCity	46350	Exoglas SL	18-May-2015
5	testapp	45923	Municipality of Bologna	18-Jun-2015
6	Citilab Test	36858	bromagosa_org	25-Jun-2015
7	Assistent per esport	30728	UPC-EET	20-May-2015
8	Web Test App	26833	asalasan_org	10-Jun-2015
9	Temperature BCN APP	26075.6667	Sala Org	09-Jun-2015
10	RunCityLondon	25287.5	Exoglas SL	19-Jun-2015
11	RunCityLondon	23248	Exoglas SL	17-May-2015
12	This is a Testing APP	22188	Developer (MY OWN)	04-Jun-2015

Table 3 List of applications with highest latency spikes over time period selected

5.2.4 Maximum Concurrency

Concurrency is defined as the number of requests being simultaneously processed at a

given time. There are several states a request can be in:

- The initial TCP connection phases,
- Request message processing in the Gateway,
- Request servicing by a back end system, and
- The sending of the response back to the originating system.

Once a message reply is sent back to the requesting system, the Gateway resources are freed to process other requests.

Concurrency is usually the most misunderstood statistic in any performance discussion.

5.3 Number of Users and Latency

5.3.1 Planning

Planning for a good user experience and sizing the iCity Platform solution is a complex undertaking as there are different parameters as inputs and many ways of looking at the problem. Doing the calculation here is important to understand the issues.

One of the most common assumptions in sizing is that large concurrency is required to support a large number of simultaneous users interacting with the application. The usual mandate is to support your user base, and to plan to accommodate a worst case situation, so let's see what real concurrency is needed by a large number of users.

5.3.2 The User Base

The following analysis is based on 20,000 users accessing a single application concurrently.

It is assumed that the application is web based, but has a core component that is sourced from some services component, i.e. the portal model. Most of the HTTP requests for a given page are things like images, CSS and other small static files and are serviced by web servers, not application servers, and so are not considered in this analysis. The calculations presented here are also applicable for fat client GUI-style applications because the same kind of technology choices around minimizing server round trips for heavyweight services also hold true for GUI applications.

5.3.3 Application Design

Once the applications are available, we will assess them and verify if they comply at least with the best practices as described below. Designing an application to perform live queries for small pieces of user interface content is not good practice whether it is a client/server, a web app, or fat client. Waiting for even local network latency to fill in the content of UI elements like drop-down lists gives extra waiting states during the displaying of the content. This make the UI appear unresponsive, and makes a large client base roll-out impractical for even unsecured applications, just from the sheer volume of the requests.

We are making a best practice assumption that services applications are designed to make one or two larger critical path requests as the core of the application service. We assume

that most pages have a single type of information the user wants to view, but some will be more complex. We assume an average of 1.25 service requests per page view, reflecting a mix of page types.

5.3.4 Service Latency

Static content requests that require no processing should be sub millisecond in latency, but actual service requests are normally in the 10 milliseconds to 5000 milliseconds range on the back end. Later we describe how the service request latency is a hugely important number in determining required concurrency.

So far we therefore have 20,000 users, with 1.25 service requests per page, and each of those request taking from 10 to 5000 milliseconds to process.

5.3.5 Requests Per Second

To make this analysis, we need to understand better the end users (mainly citizens for iCity) of the applications created.

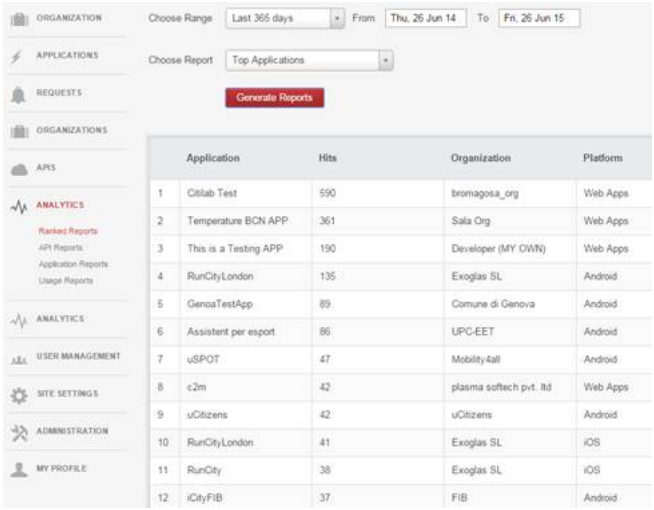
Next we need to determine how many requests those users will generate.

Given the way that people read and use applications, the bare minimum time it takes to recognize a fully rendered page or UI, find the content he/she is looking for, then choose a navigation element to initiate another request is likely to be 3 to 5 seconds. That is the bare minimum. The time that users are **not** generating new requests to back end services is called the **page dwell time**.

Dwell time on a page of something like traffic information, a purchase order or a line of business task like a shipping request is going to be longer than 5 seconds.

So, given a page dwell time between 5 and 60 seconds, over the course of an hour, 20,000 users are going to generate between 0.75 and 18 million requests, or between 208 and 5,000 requests per second. This is a reasonable number for the requests per second statistic, but leads us into the discussion of needed concurrency and how latency is by far the critical statistic.

Requests Per Page	Page Dwell Time Min (seconds)	Page Dwell Max (seconds)	Request Service Latency (milliseconds)	Requests / Second Max	Requests / Second Min	Required Concurrency Max	Required Concurrency Min
1.25	5.0	120	10	5,000	208	50	2
1.25	5.0	120	20	5,000	208	100	4
1.25	5.0	120	100	5,000	208	500	21
1.25	5.0	120	1,000	5,000	208	5,000	208
1.25	5.0	120	2,000	5,000	208	10,000	417
1.25	7.5	120	5,000	3,333	208	16,667	1,042



	Application	Hits	Organization	Platform
1	Citilab Test	590	bromagosa_org	Web Apps
2	Temperature BCN APP	361	Sala Org	Web Apps
3	This is a Testing APP	190	Developer (MY OWN)	Web Apps
4	RunCityLondon	135	Exoglas SL	Android
5	GenoaTestApp	89	Comune di Genova	Android
6	Assistent per esport	86	UPC-EET	Android
7	uSPOT	47	Mobility4all	Android
8	c2m	42	plasma softech pvt. ltd	Web Apps
9	uCitizens	42	uCitizens	Android
10	RunCityLondon	41	Exoglas SL	iOS
11	RunCity	38	Exoglas SL	iOS
12	iCityFIB	37	FIB	Android

Table 4 List of APPS with most hits against an API

5.3.6 Concurrency Calculations

The calculation for the required concurrency is as follows: 20,000 users generating 1.25 service requests per page every 5 seconds would generate, on average 20K * 1.25 * (5/60) or 30,000 requests per minute or 5,000 requests per second. We need to handle 5,000 requests every second and the service takes 10 milliseconds to handle a single request. In one second there are 100 periods of 10 milliseconds, so in each of these 10 millisecond periods we need to retire 5,000/100 or 50 simultaneous requests.

Required Concurrency=Requests per second / (1/Latency in seconds)

In our first example 5,000/(1/0.010) = 5,000/100 = 50. Of very important emphasis here is the effect of latency on concurrency. Assuming a lower page dwell time of only 5 seconds, and starting from 10 to 5,000 milliseconds service latency, the concurrency requirement jumps from only 50 simultaneous requests per second required to service 20,000 users to 16,667. At this point the performance of the system will seriously deteriorate, because at 1.25 requests per page, it would take an average of 7.5 seconds just to make the data available to render the page.

There are a large number of simplifications in this calculation but it does demonstrate that characterizing the load and the user experience has a huge impact on a prediction of required concurrency. How long will users wait for data before they decide the system is too slow?

Requests per user action also has a direct relationship to concurrency. Less clear is the effect of page dwell time. These worst case numbers reflect a given user, on average, asking for new content every 5 seconds. That may be considered as fast for most pages, unless the system has been built with lots of paging through content. Then, if what they need is on page 3, they may not wait 5 seconds to ask for new content. This can create a worst case scenario unintentionally as user acceptance testing may not accurately reflect how often people generate new requests, because the environments often are not loaded with enough data to require paging through content.

Latency Is Key

Sequence Diagram

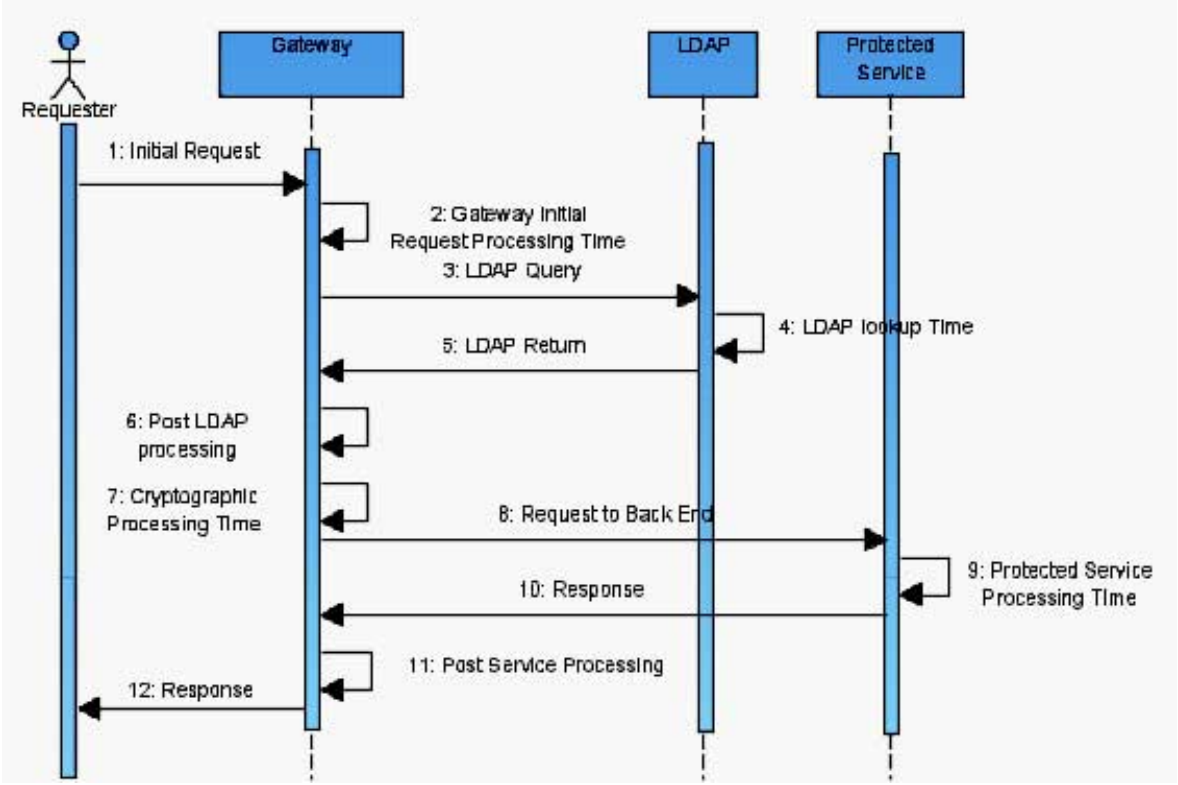


Figure 7 Sequence diagram

This sequence diagram will be used as a guideline for future measurements and will be described in a future release of this document if found relevant.

Latency is inversely proportionate to needed concurrency

In the discussion of concurrency we described an application analysis with total application service latency as a huge determining factor in concurrency requirements. There are many contributors to latency, and the iCity Gateway function needs to be monitored in that respect.

The above sequence diagram describes the processing steps and messages, internal lookup requests and points of latency when servicing a single inbound request at the Gateway function.

Some systems specifically report the time between steps 1 and 12 as the front-end response time and the time between 8 and 10 as the back end response time.

Experience has shown us that those are the two most important items to report when measuring latency.

Of note in this example is that the maximum front-end response time or more accurately, the latency experienced by the end user was only 132 milliseconds even though the back end response time was 100 milliseconds.

5.3.7 Back End Processing

In almost all scenarios you encounter in the field, **the back end processing time produces the bulk of the latency**. This is beyond our control, but the iCity Platform can help by providing: an efficient requester subsystem, controls on concurrency and connection caching for SSL. There are some components of overall latency that we end up classifying as "our local processing overhead". One of them, **LDAP Lookup Time** is minimized somewhat by our authentication cache, but still can be a limiting factor. This call to LDAP has similar analogies in Single Sign On authorizations and other methods of external decision point references. This latency is not separately described in our UI, and may in some cases result in the Gateway itself suspected as being a source of latency.

Also of particular impact is cryptography. Cryptographic operations can incur latency and/or heavy CPU usage depending on the use of internal Hierarchical Storage Management (HSM), internal software cryptography or external HSM solutions. We have very efficient cryptographic capabilities, but there is an associated mathematical complexity associated with public key operations that no system can avoid.

5.3.8 Performance Optimization

With back end latency so dominating normal performance testing, the iCity Platform should be optimized to minimize delay in back end processing.

Small messages have given typical processing rates of 20,000 requests per second, for latency in the sub-millisecond range, so in most cases; the Gateway is not contributing any significant amount to latency.

Some policy elements have latency associated and can be avoided in latency sensitive applications; Auditing is the obvious one as it has dependencies associated with synchronously waiting for the auditing subsystem to write to hard disk.

6. Summary

This report described a set of metrics and indicators as described in task 3.2 to monitor the quality and effectiveness of the proposed architecture in real deployment scenarios.

The platform is working as designed and we have had no significant technical issues.

Performance testing under high load conditions was not possible due to the lack of applications running on the platform. However, we can safely state that the cities backend system would fail a long time before the platform would experience performance issues under high load conditions and from a technical point of view, the platform will not be the bottleneck in the overall design.