D7.1.3

DEPLOYMENT OF OPEN SOURCE COMMUNITY PLATFORM

September 2013

ABSTRACT

This document describes the open source platform itself, the strategic background and the targets that should be reached in FI Content 2. An overview about the Cologne experimentation site infrastructure, an introduction in terms of setup and configuration of the "social network enabler" as well as access to the current prototype is provided.

This document is a deliverable of the FI-CONTENT 2 integrated project supported by the European Commission under its FP7 research funding programme, and contributes to the FI-PPP (Future Internet Public Private Partnership) initiative.

Disclaimer

Deliverable details

[Full project title]:	Future media Internet for large-scale CONTent experimENTation 2
[Short project title]:	FI-CONTENT 2
[Contract number]:	603662
[WP n°]:	WP7
[WP leader]:	Carmen Mac Williams, Grassroots Arts
[Deliverable n°]:	D7.1.3
[Deliverable title]:	Deployment of open source community platform
[Deliverable nature]:	Report (R)
[Dissemination level]:	Public (PU)
[Contractual delivery date]:	M6 - September 2013
[Actual delivery date]:	1 October 2013
[Editor]:	Roman Achmatow, Pixelpark
[Internal Reviewers]:	Merce Lopez, i2cat / Carmen Mac Williams & Stefan Göllner Grassroots Arts
[Suggested readers]:	
[Keywords]:	open source network community enabler
[File name]:	FI-CONTENT2_WP7-003_D7.1.3_V1.0

EXECUTIVE SUMMARY

This deliverable describes the open source platform itself, the strategic background and the targets that should be reached in FI-CONTENT 2. An overview about the Cologne experimentation site infrastructure, an introduction in terms of setup and configuration of the “social network enabler” as well as access to the current prototype is provided

More details and technical information about the platform are in the FI Content Wiki: http://ficontent.dyndns.org/

LIST OF AUTHORS

Organisation	Author
PIX	Dirk Krause, Roland Westermaier, Felix Kühn

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

AR	Augmented Reality
B2B	Business To Business
CDI	Connected Device Interface
CMS	Content Management System
DRM	Digital Right Management
DVB	Digital Video Broadcasting
EPG	Electronic Program Guide
ETHZ	ETH Zurich
FTTH	Fibre To The Home
GE	Generic Enabler
GUI	Graphic User Interface
HbbTV	Hybrid Broadcast Broadband TV
IM	Instant Messaging
IPTV	Internet protocol TV
LTE	Long Term Evolution
OTT	Over The Top
PEGI 7	Pan European Game Information
SCG	Smart City Guide
SE	Specific Enabler
STB	Set Top Box
UGC	User Generated Content
UI	User Interface
UX	User Experience

1 - Introduction – Strategic Guidelines and Targets

In the framework of the research project FI-CONTENT 2, approaches and technologies are developed that pick up the innovative character of the internet with the purpose of making this character accessible for all sorts of organisations within Europe.

The aim of the project is to deliver precise results for publicly accessible solutions in the following three areas:

- Social Connected TV
  (http://mediafi.org/applications-services/social-connected-tv/)

- Smart City Services
  (http://mediafi.org/applications-services/smart-city-services/)

- Pervasive Games
  (http://mediafi.org/a pplications-services/pervasive-games/)

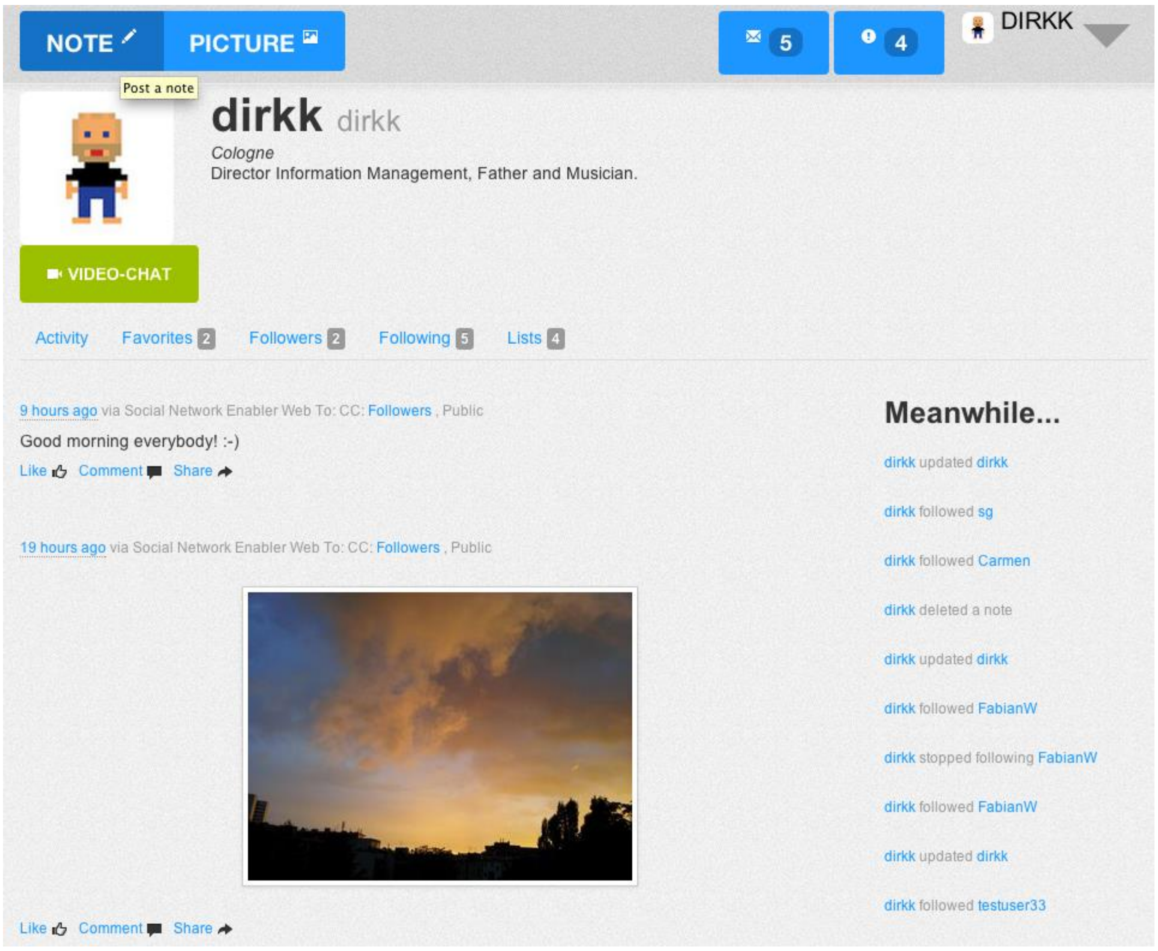
Pixelpark AG mainly operates in the area of "Smart City Services" and will develop a "Smart City Guide" prototype that will make a location specific exchange among users possible.
Therefore, the following four generic enablers, which divide into four separate software components already in use will be integrated, if technical possible:

- Identity Management
  This comprises a user administration that is focused on the security of the collected personal data of all users.

- Location
  This generic enabler contains the possibility to develop and administrate location specific contents.

- Object Storage
  Integration of cloud-based services for filing and administrating contents.

- Video Streaming
  This generic enabler allows for live video streaming and sharing of the user generated content among participants of the social network.



dirkk dirkk

Cologne

Director Information Management, Father and Musician.

9 hours ago via Social Network Enabler Web To: CC: Followers , Public

Good morning everybody! :-)

19 hours ago via Social Network Enabler Web To: CC: Followers , Public

Meanwhile...

dirkk updated dirkk

dirkk followed sg

dirkk followed Carmen

dirkk deleted a note

dirkk updated dirkk

dirkk followed FabianW

dirkk stopped following FabianW

dirkk followed FabianW

dirkk updated dirkk

dirkk followed testuser33

Figure 1 Screenshot of the SNE Prototype

2 - STRATEGIC GUIDELINES

The implementation of all ideas and considerations are based on the following guidelines:

2.1 - Exchange, integration and new contacts

The solution itself is always focused on the user and aims at promoting social interaction among users.
In order to make using the benefits of the product as simple as possible, the interaction is aimed at being as intuitive and an easy way to favour social interaction. The product will map the social network, such as, e.g. the favourite contacts of a user can be communicated more quickly and in a comprehensive way.
At the same time a certain flexibility is assured, which makes it possible to organise and restructure the contacts of a user. Thus it is possible to add, delete and reassess a contact in the same way as it is in comparable personal networks.

2.2 - Claim to innovation

The solution pursues the aim to be highly innovative. Well known features of comparable platforms are deliberately used and completed by new aspects. All innovations will offer the user an additional benefit for the sake of extraordinary user experience.

2.3 - Sharing of Contents

Users who use the product to share contents communicate within a *closed public*. This means that the content shared by a user is accessible for other users of the platform but cannot be accessed by the general public.
The overall guideline is: "Everything *can* but nothing *has to be* shared". Users can create their individual virtual identity by selecting the contents they *want* to share. This identity can be similar or partly similar to the real identity, or it can be completely different. This product places the responsibility for creating a virtual identity with the user, who can follow his own wishes.

2.4 - Positive Interaction

The product is meant to promote positive interaction in the sense of a fair and constructive communication. Therefore, users can highlight contents they liked especially (e.g. by clicking on a "Like"-button), but not those they did not like. The aim is to favour a constructive exchange on shared contents that makes intervention by a third party seem rather unnecessary.

2.5 - Open vs. Closed User Group

The product is mainly designed for the closed public mentioned in chapter 2.1 -. In a longer perspective the group of users will be widened *by the users* themselves. This means that users for whom the platform is highly relevant will be able to add new users to the platform for whom it will probably also be of considerable importance. These new *users of the second generation* will find a product that meets their needs and preferences. This makes it possible to avoid scattering losses.
The solution will not be accessible for an open user group to restrict the administration and maintenance effort to an acceptable level.

2.6 - Possibilities for extension

The product will be implemented by using a scalable system that can be extended with new functionalities in a flexible way. This makes it possible to include demands that are identified at a later stage.

2.7 - Possibilities for testing

The product is going to be implemented in separate packages that can be tested in order to get feedback as quickly as possible. The feedback will influence the further development of the product.
Thus it will be possible to identify functionalities rejected by the target users already during the developing process. The rejection of a part of the system will knowingly be accepted and shall not impede the development of new approaches.

The current prototype can be visited at http://sne.dyndns.info

3 - Product Vision and Target Definition

For the implementation of the product the following targets have been defined:

- **Product vision Smart "City" Guide**
  In a first field test the product will be tested as a virtual city guide. This includes among other features and functionalities like the integration of location specific information, which positively differ from competing products in terms of quality and relevance because they were gathered by a closed user group.
- **Integration of GEs**
  The generic enablers “identity management”, “location”, “object storage” and “video streaming” will be added to the product in a beneficial way. The exact design will be defined during the project development and is related to the technical possibilities and restrictions of the framework used for the implementation.
- **Intuitive user interface**
  To make the first access for users as simple as possible, the design of the user interface is similar to the UX-standards known and mastered by the users through their habits of using websites and smartphone apps. All elements of the user interface are developed in a way that they can easily and intuitively be used.
- **Integration of mobile devices**
  In the first field test the product will mainly be used with mobile devices. Therefore, the product will use a responsive layout grid which supports the flexible display of contents on different end devices. The following end devices are taken into account:
  - **Smart phone:** With regard to the size of the display and the specialities for usage related to it, the main core functions will be displayed. Functions and information of less importance will get a lower priority and can even be omitted. Thanks to a fluid grid smart phones with different display sizes and resolutions can be used.
  - **Tablet PC:** These kind of devices have a considerably larger display. The amount of information displayed on a tablet can therefore be much larger than on a smart phone, but other restrictions arouse from the fact that it is operated via touch screen.
  - **Desktop computer:** In this case the usage concept is transferred to a mainly stationary kind of device. The big display, keyboard and mouse enable the user to operate more precisely, thus additional functions could be added in this case.
- **Prototypic implementation of a field test**
  The product will be given to the testers at a prototypic stage. Any divergences or aspects that have not fully been completed but do not interfere with the functionality of the product will be consciously accepted.

4 - EXPERIMENTATION SITE AND INFRASTRUCTURE (OVERVIEW)

The Cologne Experimentation site infrastructure is an open source community platform available for user community management and to develop, test and evaluate novel Smart City Services, Gaming and social TV applications including specific enablers from FI-CONTENT 2 and Generic Enablers from FI-Ware.

This component is responsible for social features like:

- user registration and sign-in
- user profile
- text status updates
- picture upload status and status updates
- etc

4.1 - Technical overview of the open source community platform

Software Stack

As a supporter of the open web technologies the preferred stack comprises:

- HTML5
- CSS3
- JavaScript

On the client side, and the MEAN stack on the server side:

- MongoDB
- Express
- AngularJS
- NodeJS

Components

The Social network Enabler consists of different components, which are loosely coupled through Ajax and REST calls (SOA). Also we aim towards an exchangeability of components via standard protocols. In theory you can exchange each component though the mileage may vary.
To support this we intend to support this exchangeability by a set of instructions (configuration standards, etc) that ease this task. The fragmentation to several services is intended because we try to nurture exchangeability for different business models.

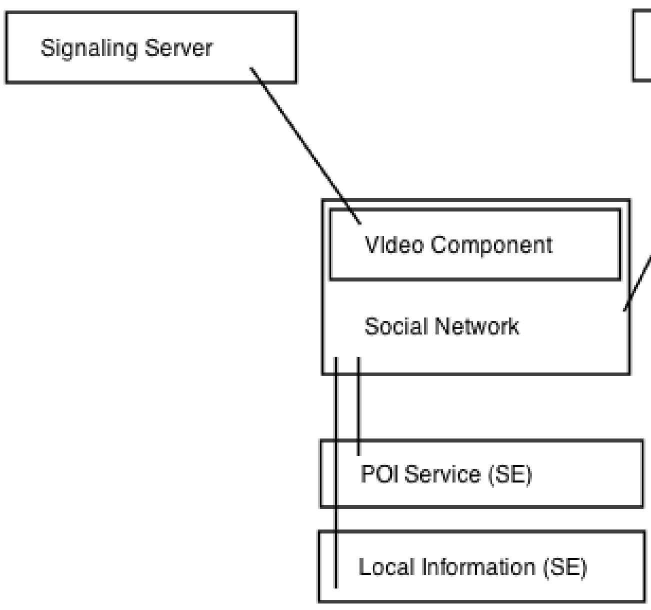


Figure 2 Social Network Enabler

Social Network Enabler

This component is responsible for social features like:

- user registration and sign-in
- user profile
- text status updates
- picture upload status and status updates
- etc

This software component will be the pump.io server: http://pump.io/. This is a federated social server software.

Signaling Server

For simple messaging purposes a signaling server based on socket.io will be used. This component is used by the other components as a messaging server with a simple state machine.

Namely the first service to use this will be the video component, which will be implemented on the client side via WebRTC.

External Components

External components are systems to provide Point Of Interest (P.O.I.) information and recommendation engines. Intended components are:

- POI Service SE (if applicable)
- Local Information SE
- Recommendation Engine SE (if applicable)
- Identity Management GE
- Video Streaming GE

The FI CONTENT Specific Enablers are owned by different partners. Therefore, different terms and conditions might apply.

5 - OVERVIEW OF THE SOCIAL NETWORK SE

Social Network SE Core

- REST Webservice
- Web interface via HTTP

Intended Audience

This specification is intended for both application developers and service providers. For the former, this document provides a specification of how to interoperate with the Social Network SE API. For the latter, this specification indicates the interface to be provided in order for clients to interoperate with services which benefit from the provided functionality. To use this information, the reader should firstly have a general understanding of the social network features. You should also be familiar with:

- RESTful web services
- HTTP/1.1
- OAuth 1.0

Additional Resources

You can download the most current version of this document from the FIcontent platform documentation website at the FI-CONTENT Common Platform - Release 09/13 (http://ficontent.dyndns.org).
For more details about the usage of the Social Network SE within the FI-CONTENT platforms, please refer to the FI-CONTENT Architecture.

For more details about getting started with the Social Network SE within your own application or service, please refer to the FIcontent.Common.SocialNetwork.DeveloperGuide as a first starting point.

Parts of this document are derived from pump.io documentation.
(https://github.com/e14n/pump.io/blob/master/API.md).

5.1 - Overview of the Social Network SE

The main functionalities that the Social Network SE provides are:

- User authentication (vs. GE)
- User participation and collaboration
- User generated content

The Social Network SE is a combination of interconnected services to provide above functionalities described above. As such it is a construction of open source software solutions with their respective documentation found on the open source directory Github. This is especially true for the specification below, which is essentially based on the pump.io documentation.

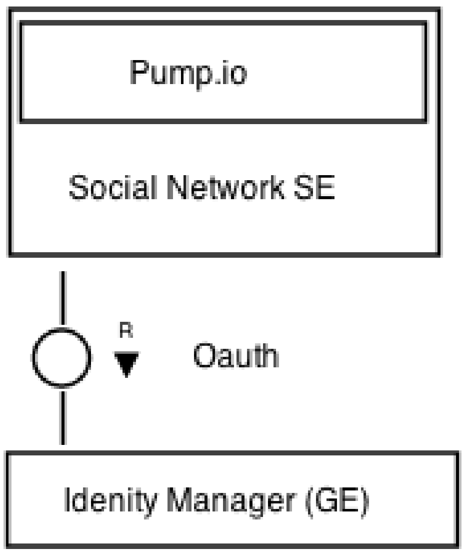


Figure 3: FMC Diagram

5.2 - Installation and configuration

The repository of the Social Network Enabler is on Github. The installation and configuration guides of the components are maintained and referenced there.

6 - Technical Description of Activity Streams

Activity Streams is a format for representing events or activities in a social network or in collaborative software. It's a JSON format, meaning that on the wire data looks like JavaScript literals.

Activity Streams data includes a few different kinds of things:

- objects: These represent real or digital objects. Every object has an `objectType` property; some common object types are "person", "note", "image", "place". There are some standard properties that all object types support, like `displayName` for an easy-to-use short text name, or `content` for HTML content representing the object. Some types have unique properties, like a "place", which has a `position` and an `address`. Every object has a unique `id`, which is a URI that identifies that object globally.
- activities: An activity is something that happened. It's like a sentence; it has an `actor`, who did the thing, a `verb` that is what happened, and an `object` which is what it happened to. Activities can also have some other properties, like a`location` or a `target`. Activities also have an `id` property that uniquely identifies the activity.
- collections: These are ordered lists of activities.

An example of an Activity Streams activity:

```
.    {
.        "id": "http://coding.example/api/activity/bwkposthw",
.        "actor": {
.            "id": "acct:bwk@coding.example",
.            "displayName": "Brian Kernighan",
.            "objectType": "person",
.            "url": "http://coding.example/bwk"
.        },
.        "verb": "post",
.        "object": {
.            "id": "http://coding.example/api/note/helloworld",
.            "content": "Hello, World!"
.            "objectType": "note"
.        },
.        "published": "1973-01-01T00:00:00"
.    }
```

This activity has an `id` to uniquely identify it, an `actor`, a `verb`, and an `object`. It also has a publication timestamp,`published`. The actor is a "person" with a name and an `id` that is
an "acct:" URI, as well as an `url` of a profile page. The object is a "note".

The activity streams specification is long; there are also several extensions that pump.io supports. The Activity Base Schema lists some common object types and verbs.

It's possible to make new object types and new verbs by using full URIs for the `objectType` or `verb` property. Unknown object types or verbs will be stored but won't cause side-effects.

6.1 - Feed basics

Each user account on a pump.io server has two main feeds:

- An *activity outbox* at `/api/user/<nickname>/feed`. This is where the user posts new activities, and where others can read the user's activities.
- An *activity inbox* at `/api/user/<nickname>/inbox`. This is where the user can read posts that were sent to him/her. Remote servers can post activities here to be delivered to the user (see below).

The feeds are collections of activities.

6.2 - Creating an activity

To create a new activity, a client posts the activity in JSON format to the user's feed. The Activity Pump will automatically add IDs where needed and the user's profile as an `actor`.

Here is an example of HTTP request to create a new activity:

```
.     POST /api/user/bwk/feed HTTP/1.1
.     Host: coding.example
.     Authorization: OAuth oauth_consumer_key="[...]",
.         oauth_token="[...]", [...]
.     Content-Type: application/json

.     {
.         "verb": "follow",
.         "object": {
.             "id": "acct:ken@coding.example",
.             "objectType": "person"
.         }
.     }
```

Note that the request uses OAuth authorization to authenticate the user. The only form of authentication allowed for the activity outbox is OAuth; the user must authorize the client before new activities are created.

The HTTP response to this request will be a fully-defined activity with all the IDs and timestamps filled in.

```
.  HTTP/1.1 200 OK
.     Content-Type: application/json
.     {
.         "id": "http://coding.example/api/activity/bwkflwken",
.         "actor": {
.             "id": "acct:bwk@coding.example",
.             "objectType": "person",
.             "displayName": "Brian Kernighan"
.         },
.         "verb": "follow",
.         "to": [{
.             "id": "acct:ken@coding.example",
.             "objectType": "person"
.         }],
.         "object": {
.             "id": "acct:ken@coding.example",
.             "objectType": "person",
.             "displayName": "Ken Thompson"
.         },
.         "published": "1974-01-01T00:00:00",
.         "links": [
.             {"rel": "self", "href":
"http://coding.example/api/activity/bwkflwken"}
.         ]
.     }
```

6.3 - Side-effects

Posted activities may have side-effects; in the above case, the actor "bwk" has followed another person, "ken", so that activities that "ken" shares with his followers will also go to "bwk"'s inbox.

Most activity verbs *don't* have side-effects. In this case, the pump.io will just store the data about the activity, and distribute the activity according to the social graph, but it won't change the state of that graph.

Verbs that have side-effects include:

- "post": Creates the object.
- "update": Modifies the object, if it exists, to have the new structure in this activity.
- "delete": Deletes the object. After a delete, only a shell of the data about the object will remain.
- "follow": Makes the actor follow the object. After a follow, activities that the object shares with his/her followers will go to the actor's inbox, like a subscription. The actor is added to the object's followers collection, and the object is added to the actor's following collection.
- "stop-following": Makes the actor stop following the object. After this point, activities that the object shares with his/her followers will not go to the actor's inbox, but any existing activities will stay there.
- "favorite" or "like": These are synonyms. The actor is added to object's list of "likers", and the object is added to the actor's list of favorites. "unfavorite" or "unlike": Undoes a "like" or "favorite".
- "add": If the target is a collection that belongs to the actor, will add the object to the collection. Good for adding users to user lists.
- "remove": If the target is a collection that belongs to the actor, will add the object to the collection. Good for removing users from user lists.

Other verbs may have side-effects in the future, especially the ones around friendships, groups, events, and playing media.

6.4 - Reading collections

Reading from the activity outbox or activity inbox requires OAuth authentication. The activity inbox requires user authorization; you can request data from the activity outbox using plain old 2-legged OAuth client authentication if you want.

6.4.1 - Arguments

Collection URLs can have arguments that change the size of the
collection. By default, the collection returned will include only the
most recent activities -- usually 20.

Collection URLs take the following parameters:

- *count*. The number of items to return (default is usually 20). This maxes out at 200, usually.
- *offset*. Zero-based offset specifying where in the collection you want to start. Default 0. This is a bad way to do pages, since activities are added at the beginning of the collection.
- *before*. An activity ID. Will get activities that went into the collection immediately before the specified activity (not inclusive). A good way to "scroll back" in a collection.
- *since*. An activity ID. Will get activities that went into the collection immediately after the specified activity (not inclusive). A good way to get what's new in a collection since you last polled it.

6.4.2 - Navigation links

Collection objects include links to help with navigation, using the Multi-page Collections schema.

Note that because activities are in reverse chronological order, understanding what's "next" or "previous" is kind of unintuitive. We assume you start with the current activities and scroll backwards in time, so "older" stuff is "next" and "newer" stuff is "prev".

- *next* The next _older_ group of activities.
- *prev* The next _newer_ group of activities. This is provided even

if you're looking at the newest activities; it makes it easier to
just get the most recent stuff you haven't seen.

6.5 - Addressing activities

pump.io supports the Audience Targeting for JSON Activity Streams (http://activitystrea.ms/specs/json/targeting/1.0/) extension, which lets you add addresses to an activity to show to whom the activity is directed.

Address properties for an activity include `to`, `cc`, `bto`, and `bcc`. The properties are arrays [] of objects {}. The objects should include an `id` and `objectType` property; they may include other properties.

Activities that are `to` a user or list will go into their "direct" inbox (see below).
`bto` and `bcc` properties won't be shown to any users except the author.

pump.io uses the addresses for three things:

- *Delivery of activities*. Depending on the addresses, the activity will be delivered either to local user inboxes or to remote users (see below!).
- *Access control to activities*. Requesting an activity directly from its REST endpoint will give a 403 error. Also, feeds are filtered to only show activities that were addressed to the requester.
- *Access control to objects*. Access to objects generally depends on if the requester was a recipient of the "post" activity that created it. The REST endpoint for an object (see below) will return a 403 status code otherwise.

7 - Social Network SE API Specification

Since the Social Network enabler is based on pump.io, it provides the same REST (Representational State Transfer) API with following specific endpoints.

7.1 - Types of address

There are four types of address supported:

- *The public*. "The public" is an object with `objectType` equal to "collection" and the special ID "http://activityschema.org/collection/public". Activities addressed to the public will be delivered to all followers, and will be visible to anyone -- even unauthenticated users.
- *Followers*. A user's own followers can be addressed with an object with `objectType` equal to the follower stream URL of the user – usually`http://<hostname>/api/user/<nickname>/followers`. Activities addressed to followers will be delivered to all followers, and will only be visible to followers.
- *Users*. Users can be addressed by their profile object -`objectType` is person, and `id` is something like `acct:<nickname>@<hostname>`.
- *Lists*. Users can create collections of people or other objects. An address with `objectType` "collection" and the ID of one of the actor's own collections will result in delivery to the members of that list, and members of the list will be allowed to view the activity and/or object.

7.2 - Default addresses

Default addresses are added if an activity is posted to the activity outbox with no address properties. We try to be reasonable with the defaults, as follows:

If an activity has an object that is `inReplyTo` another object, the addresses for the "post" activity of the original are used.
If an activity is an "update" or a "delete" of an object, the addresses for the "post" activity of that object are used. If an activity has an object that is a "person", the person is added as a `to` address. Otherwise, the actor's followers are added as a `cc` address.

These defaults will probably change over time; if you want to make sure that specific addresses are used, you should definitely add them explicitly.

7.3 - Major and minor feeds

Some activities are more important than others. pump.io provides sub-feeds of the outbox and inbox, divided by whether the activity is "major" or "minor". Roughly speaking, posting new content is "major", and changes to the social graph or reactions to other content are "minor".

The feeds are at `/api/user/<nickname>/inbox/major`, `/api/user/<nickname>/inbox/minor`, `/api/user/<nickname>/feed/major`, and `/api/user/<nickname>/feed/minor`.

The major and minor feeds will respond to POST requests. You can only post major activities to the major feed and minor activities to the minor feed.

7.4 - Direct inbox

Activities that have a `to` or `bto` property that includes the user's address will be listed in the user's "direct" inbox. There are major and minor variations of this inbox, also.

The feeds are at `/api/user/<nickname>/inbox/direct`, `/api/user/<nickname>/inbox/direct/minor`, and `/api/user/<nickname>/inbox/direct/major`.

7.5 - Object endpoints

When objects like a "note" or an "image" are created, they're assigned a REST endpoint, usually something like `http://<hostname>/api/<objectType>/<id>`, where the `<id>` is a random value: it's a UUID in URL-safe base-64 format.

You can get the object endpoint from the object's `links` property; it's the link with `rel` value `self`.

Objects respond to HTTP GET requests with an Activity Streams JSON representation of the object.

The author of an object can PUT to the object endpoint; this will update the object; it will also generate an "update" activity.

The author of an object can DELETE to the object endpoint; this will delete the object. It will also generate a "delete" activity. Sending a GET to an object endpoint for an object that was deleted will return a 410 Gone status code

7.5.1 - Links

Objects also have a `links` property; it's an array of objects.

- *self*. The canonical, true, real, one-and-only for-sure HTTP endpoint to retrieve an Activity Streams JSON representation of the object

7.5.2 - Collection properties

Objects have related collections, as defined in the misnamed
[Responses for Activity Streams](http://activitystrea.ms/specs/json/replies/1.0/). These
particular properties are probably interesting:

- `replies`. Objects that were posted with an `inReplyTo` value of this object.
- `likes`. People who have sent a "favorite" activity with this object as the `object` property.

In representations, these collections will use have the first ~4 items included in the `items`.

"person" objects have these collections instead.

- `followers`. People who follow this person.
- `following`. People who this person is following.

- `lists`. (non-standard) Lists that belong to the user.
- `favorites`. (non-standard) Objects that the user has sent a "favorite" activity about.

A user can follow someone else by posting to their `followers` collection. They can also add a favorite by posting the object to their `favorites` collection. Both of these will generate the appropriate activity with default addresses.

"collection" objects have this property:

- `members`. The collection of members of the collection.

7.6 - Activity endpoints

Every activity also has a REST endpoint, usually `http://<hostname>/api/activity/<uuid>`.

It will respond to a GET with the JSON representation of the activity.

The REST endpoints also respond to PUT or DELETE requests. These won't cause side-effects, however; deleting a "follow" activity won't change the list of followers. It's probably much better to post another activity, such as "stop-following", to reverse the effects of an
activity.

7.7 - Authentication

Almost all API endpoints require OAuth authentication; most of them require user authorization. The OAuth sign-up flow is straightforward, with the following endpoints:

- `/oauth/request_token` to get an OAuth request token
- `/oauth/authorize` to authorize an OAuth request token
- `/oauth/access_token` to turn a request token into an access token

To get profile data on the authenticated user, use the endpoint at:

- `/api/whoami` - returns an activity object for the registered user. Uses a redirect to the canonical endpoint, so you should follow that.

7.8 - 2-legged OAuth

The following endpoints only require 2-legged authentication; you don't have to get user authorization or provide an `oauth_token` parameter. However, getting a user authorization will allow getting some stuff that's otherwise private:

- activity outbox
- followers collection
- following collection
- favorites collection

7.9 - Client registration

You can request a new client ID for OAuth authentication automatically, using [OpenID Connect Dynamic Client RegistrationNote.
The registration endpoint is at `/api/client/register`. You can also discover it in the host-meta file with link-rel "registration_endpoint".

The client registration will accept some of the parameters that OpenID does. Here's what it supports:

- *type*: one of the values `client_associate` (when registering a client) or `client_update` (when updating the details of a previously-registered client)
- *client_id*, *client_secret*: only when `type` is set to `client_update`.
- *contacts*
- *application_type*
- *application_name*
- *logo_url*
- *redirect_uris*

7.10 - Dialback authentication

If you use Dialback Access Authentication when requesting an OAuth client identity, the client ID will be associated with the host or webfinger that you use for authentication.

You can then make 2-legged OAuth requests (no oauth_token) to other parts of the API, with the authorization of that webfinger or host.

This currently only works with remote delivery (see below), but probably other parts of the API will support it in the future.

7.11 - User registration

There is a collection of all users at the endpoint `/api/users`. To create a new user, POST a user representation (see below) to the list. You can also get the latest registered users by GETting the collection.

The JSON object representing the user has the following properties:

- `nickname`: The user's nickname. 1-64 characters, including only ASCII capital and lowercase letters and numbers as well as "-", ".", and "_". The nickname is immutable and unique per server; it can't be changed.
- `password`: The plain-text password. This isn't returned when you GET the user object, but you have to provide it when registering or updating the user.
- `profile`: a "person" object. This is created automatically when you create a new user; don't try to add it yourself. Don't update this directly; update the person through its object endpoint.

7.12 - User objects

Each user has an HTTP endpoint at `/api/user/<nickname>`. It's useful to GET the user representation or to get the profile for a user.

The user can PUT a new representation; since `nickname` and `profile` are immutable, this is pretty much only useful to change your password.

The user can DELETE the endpoint; it will delete the user account, but not much else. In particular, it won't clean up all the user's activities, profile, followers, following, lists, or published objects. This is an intrinsic behaviour of a social network, since every information can be referenced, linked or answered somewhere else. This is not in contrast with the ethical requirements since the test user group will have aliases that cannot be traced back to the real person.

To eternally delete the information of a specific an application can be scripted that connects to the SNE and traverses through the object database and deletes the specific records.

7.13 - Discovery

pump.io supports Web Host Metadata (http://tools.ietf.org/html/rfc6415) to discover information about users and hosts.

It supports the XRD and JRD versions of the discovery output; usually JRD is preferred.

For hosts, we provide these link-rel types:

- `lrdd` - to find the endpoint for per-user discovery
- `registration_endpoint` - to find the endpoint for client registration
- `dialback` - to find the endpoint for Dialback authentication verification for this host

For users, we support only `<nickname>@<hostname>` ID format. These link-rel types are provided:

- `http://webfinger.net/rel/profile-page` - to find the profile page
- `activity-inbox` - the user's activity inbox
- `activity-outbox` - the user's activity outbox
- `dialback` - for Dialback verification for this user

7.14 - Remote delivery

It's possible to deliver activities from a remote host to a user's inbox by POSTing to the inbox. You need to use 2-legged OAuth authentication, and the client ID used for the OAuth has to be associated with the webfinger ID of the actor who did the activity.

When activities are posted to a user's activity outbox with addresses of remote users, pump.io tries to deliver them using this method. In rough outline:

1. It tries to discover an `activity-inbox` link for the person.
2. It tries to discover a `registration_endpoint` for the hostname part of the activity-inbox endpoint.3. It registers new OAuth credentials for the sender with the remote host.
   It POSTs the activity to the person's activity inbox endpoint, with the OAuth credentials.

8 - CONCLUSION

This deliverable has described the activities to develop and run the open source platform and the targets that should be reached in WP7 of FI-CONTENT 2. It gives an overview about the main features of Cologne experimentation site infrastructure and the “social network enabler”.