Ecosystem infrastructure for smart and personalised inclusion
and PROSPERITY for ALL stakeholders

D204.1 Modularized media transformation software platform to enable community/crowd improvement of code

Project Acronym	**Prosperity4All**
Grant Agreement number	**FP7-610510**
Deliverable number	**D204.1**
Work package number	**WP204**
Work package title	**Modularized media transformation software platform to enable community/crowd improvement of code**
Authors	**Dean Jansen, PCF**
Status	**Final**
Dissemination Level	**Public/Consortium**

Delivery Date **28/7/2016**
Number of Pages **17**

Keyword List

Conversion, modularization, and documentation

Version History

Revision	Date	Author	Organisation	Description
1	12/07/2016	Dean Jansen	PCF	Initial Draft
2a	14/07/2016	Doris Janssen Matthias Peissner	FhG-IAO	Internal Review: only minor suggestions: 1. Explanation (few words) of the contribution to the global architecture of P4A 2. Some more statements about the background, what Amara is and esp. Unisubs (state before the P4A work on it started) 3. Add some screenshots / illustrating pictures about the work you've done, not only links
3	22/07/16	Dean Jansen	PCF	Response to internal review feedback.
4	25/07/16	Marco Aimar	ENG	Pre-submission feedback: clearer introduction to Amara features
5	27/07/16	Dean Jansen	PCF	Response to Marco Aimar's pre-submission feedback.

Table of Contents

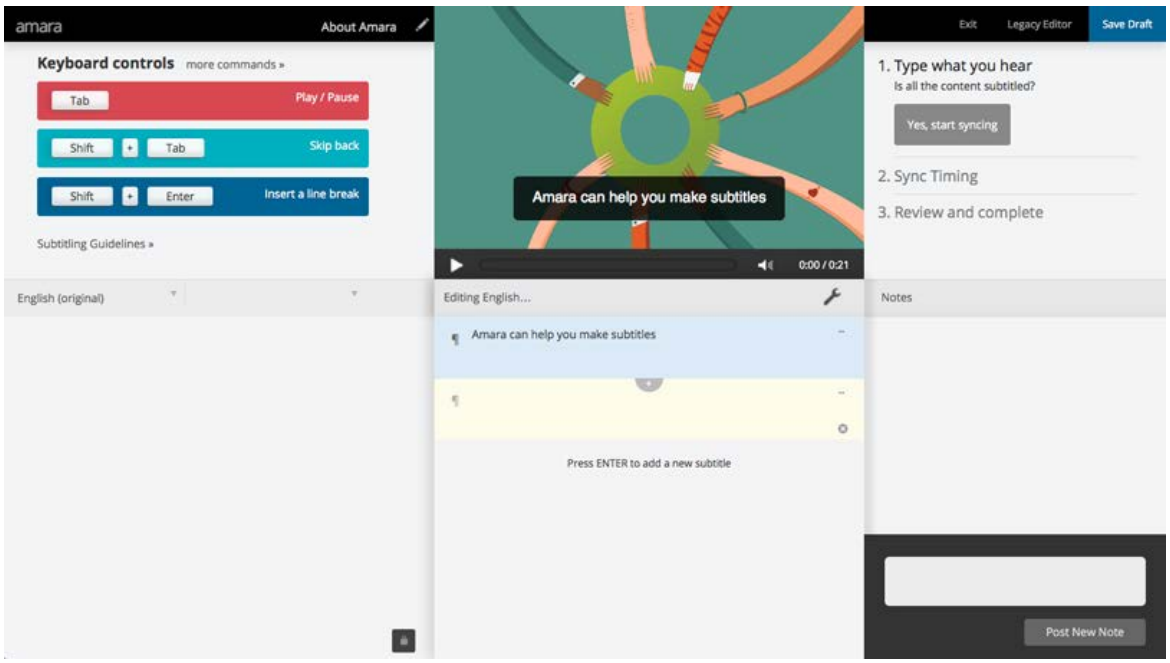
List of Figures

Executive Summary

The Participatory Culture Foundation (PCF) hosts and develops Amara, a platform for crowd/community created captions and translated subtitles. Amara is undergirded by the Unisubs Engine, an open source platform which will be included as an infrastructural building block in the P4A's DeveloperSpace.

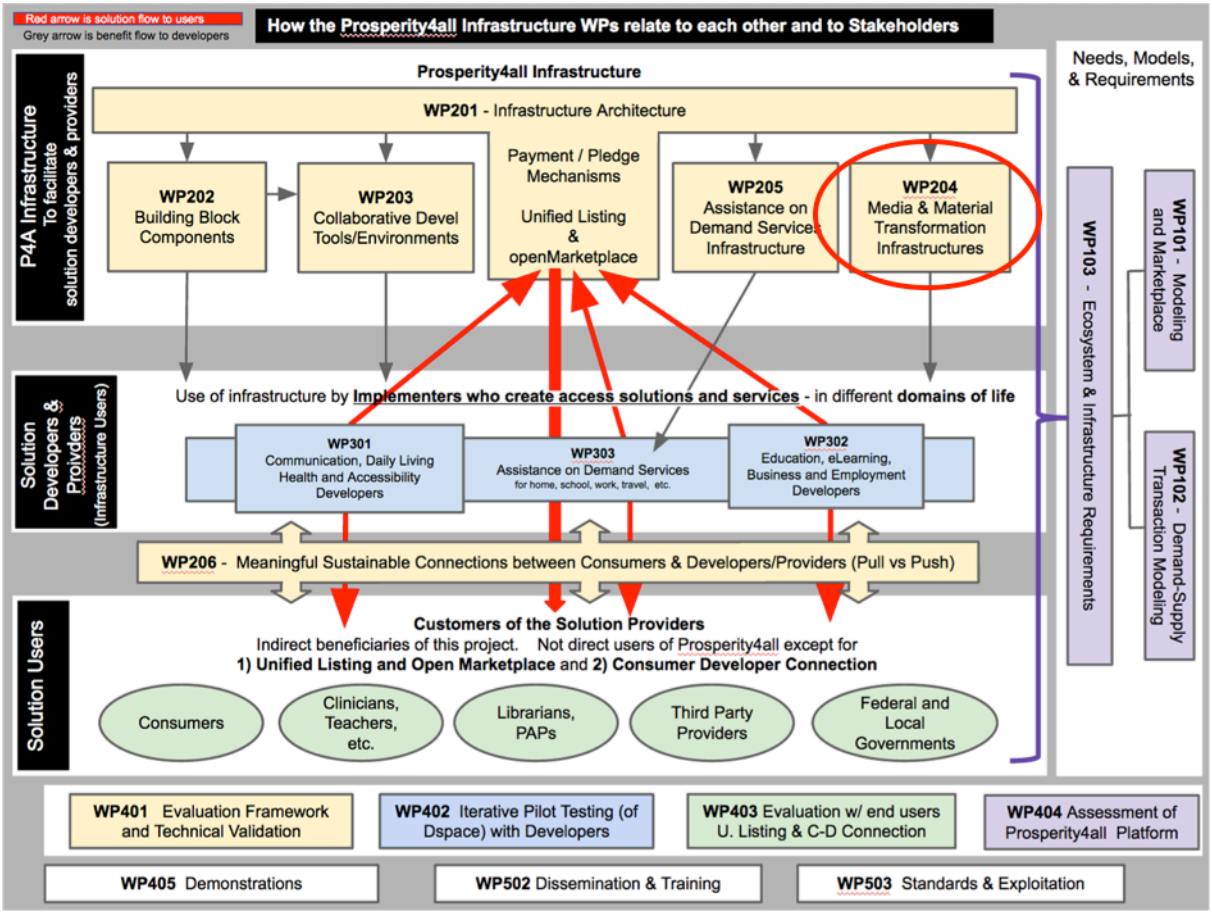
This document provides a summary of enhancements made to the Unisubs Engine that will facilitate easier development and enhancement of the software by developers, organizations, researchers, or other interested parties.

Figure 1: Amara Caption & Subtitle Editor



1 Contribution to the Global Architecture

Figure 2: Overall Picture of Prosperity4all, with WP204 Highlighted



The Unisubs Engine is a module available in the P4A Infrastructure; the platform enables communities to caption and/or translate videos for greater accessibility. The source code for the Unisubs Engine will be made available through P4A's DeveloperSpace platform.

Prior to being part of the P4A consortium, PCF launched Amara as a prototype, which evolved into an award-winning platform for community-driven captions and subtitles. Although the underlying software, the Unisubs Engine, was open source since the beginning, it was not particularly modular or easy to extend.

The goal, in improving the Unisubs Engine as a member of the P4A consortium, is to increase the platform's flexibility, modularity, compatibility, and ease-of-use – ultimately driving broader and more cost-effective accessibility for video. The improvements to the Unisubs Engine will be made as improvements to the code and documentation (as in D204.1), as well

as additional features. The P4A-driven features will add compatibility with additional video services, enable users on mobile devices to participate in community accessibility efforts, and enable community members to work together in real time to caption and subtitle video.

2 Introduction

Amara is a project of the Participatory Culture Foundation (PCF), a not-for-profit organization. Amara's mission is to ensure that all online video content is accessible to everyone regardless of hearing ability or language barriers. At the heart of the Amara platform is an award-winning web-based caption and subtitle editor that makes it easy for users to create and/or improve captions, subtitles, and translations in a collaborative Wikipedia-style fashion. Amara helps large communities coordinate the work of making video accessible on a global scale.

The purpose of the T204.1 enhancements in WP204 is to make the Unisubs Engine –which is the open source software and platform that undergirds Amara– more attractive to potential engineers, developers, and researchers. To accomplish this task, three key areas of the software were reviewed: and plans were laid, and subsequently executed, for improving the infrastructure, systems architecture, and documentation. The goal is to make the platform easier to understand, run, build upon, and extend.

The first key area reviewed was infrastructure – in other words, the technical infrastructure and underlying software components that support the Unisubs Engine. The goal was to move away from a monolithic virtual machine for development and separate services on production to a containerized environment that was easy to setup, simple to run, automated, and consistent between production and development.

The second key area examined was code modularity. The goal of these changes is to make the Unisubs Engine more readily extended and adaptable. As initially imagined, the platform had very little room for modular extensibility. Modifications to functionality were scattered into the codebase and were difficult to maintain; with greater modularity, this limitation could be overcome. Developers can now more easily build improvements independently of the Unisubs Engine and customize specific areas of the platform for disparate use cases.

The final key area evaluated was the platform's documentation, both in-code, as well as other areas where the Unisubs Engine's documentation was publicly presented. In earlier days, the documentation was somewhat piecemeal, irregularly/inconsistently applied, and non-systematized. Today there is a system for standardized documentation that is consistent and automatically formatted and presented to the public in an easy-to-follow format.

These three areas of improvement enable the Unisubs Engine to be more easily set up and modified by anyone interested in community-driven software for video accessibility.

3 Conversion, Modularization, & Documentation

For each of the three primary areas of focus for D204.1 –Conversion, Modularization, & Documentation– the initial situation will be analyzed, goals and targets for improvements will be reviewed, and, finally, the actual implementation will be summarized.

3.1 Conversion: Infrastructure & DevOps

3.1.1 INITIAL SITUATION

The Unisubs Engine relies on a cluster of infrastructural building blocks – pieces of open source software that include: a web server, a database, a message broker, a search engine, and so on. To reduce complexity in installation and management of these components, the Unisubs Engine has relied on server virtualization. The team's initial approach to virtualization –which relied on software packages such as Vagrant and Puppet– were cumbersome, complex, and required significant resources to run/maintain/manage. This was a barrier to entry for anyone wishing to run the Unisubs Engine, whether to offer it as a service on the web, to develop new features, or simply for exploratory purposes.

The Unisubs Engine's installation process required a significant time investment before one could truly engage with the code – finding ways to reduce this barrier was critical.

3.1.2 GOALS & TARGETS

To minimize the burden of installing the Unisubs Engine, the team reviewed various strategies for streamlining the installation process: ensuring consistency, stability, and ease of use. One particular school of thought, known as DevOps, proved useful in guiding decisions aimed at effectively streamlining the development, deployment, and maintenance of the Unisubs Engine[1].

DevOps is a philosophy, the goal of which is summed up well by Gene Kim in the IBM developerWorks series, *DevOps distilled, Part 1: The three underlying principles*:

> DevOps results in the fast flow of planned work (for example, high deploy rates) while simultaneously increasing the reliability, stability, resilience,

[1] https://en.wikipedia.org/wiki/DevOps

and security of the production environment[2].

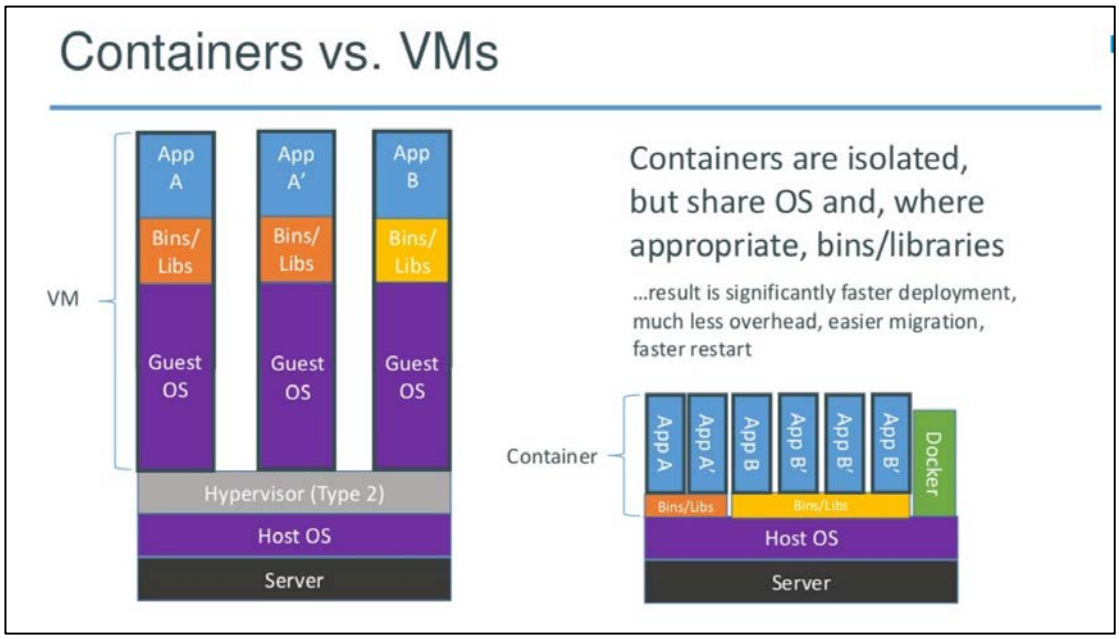
A key goal for converting the Unisubs Engine was to find an approach that would enable developers to quickly deploy and iterate on the software. With DevOps as a guide for transforming the platform, the product team undertook a major overhaul to the underlying structure of the Unisubs Engine.

After a careful review of various approaches to restructuring the Unisubs Engine, the team decided to focus on converting the infrastructure to *operating-system-level virtualization* and *containerization*[3]. The primary goal of containerization, in our case, was to efficiently ensure consistency and predictability in server builds[4].

Modern containerization software allows a code stack to be deployed rapidly with low overhead and high consistency. Not only does this guarantee that the software will always run the same, regardless of environment, but it also simplifies scalability and stability in production deployments. These benefits are possible because each container has its own isolated environment which can be copied and reproduced between machines. OS-level virtualization is utilized to provide this without sacrificing speed – in fact, it typically has lower overhead than traditional virtualization techniques. The differences can be summarized in Figure 2.

[2] http://www.ibm.com/developerworks/library/se-devops/part1/part1-pdf.pdf
[3] https://en.wikipedia.org/wiki/Operating-system-level_virtualization
[4] https://www.digitalocean.com/community/tutorials/the-docker-ecosystem-an-overview-of-containerization

Figure 3: A visual comparison of Containers and Virtual Machines[5]



The team decided to use Docker, which is both the open source and de-facto standard for containerization[6]. Docker, which was initially released in 2013, is very actively developed and has a broad and dedicated following. Contributors to the platform include engineers from: the Docker team, Red Hat, IBM, Google, Cisco Systems, and more[7]. Docker has cultivated a healthy and robust development ecosystem, which indicates that the platform is likely to continue evolving, improving, and being well supported for many years to come.

3.1.3 IMPLEMENTATION

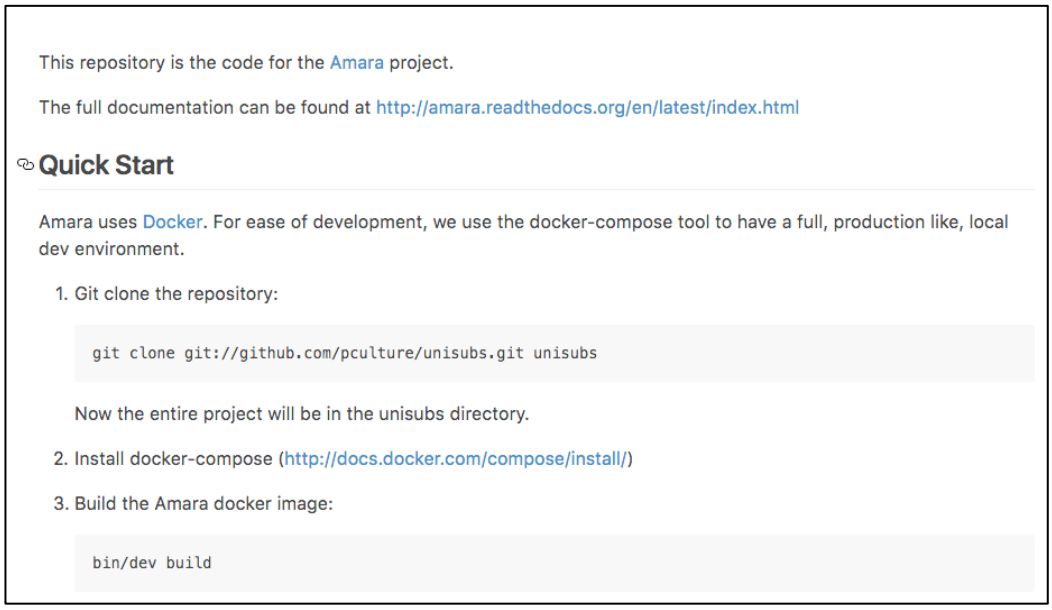
By using Docker to containerize the Unisubs Engine itself, as well as related dependencies – MySQL, RabbitMQ, Memcached, HAProxy, and so on–, the entire software stack has become quicker to get up and running. This ultimately means less time spent installing various components and subsystems, which gives a developer more time to explore, code, and/or actually run the Unisubs Engine as a web application.

In addition to simplifying installation and management, the conversion to Docker has yielded additional benefits for production usage in terms of uptime availability and scalability. When

[5] http://www.slideshare.net/dotCloud/docker-intro-november
[6] http://www.networkworld.com/article/2226751/opensource-subnet/docker-becomes-de-facto-linux-standard.html
[7] https://en.wikipedia.org/wiki/Docker_(software)

a deployment of the Unisubs Engine requires additional computational resources, additional *worker containers* can quickly be spun up to meet higher demand. Docker's lightweight virtualization architecture also makes it easy to hot swap in updated components, so that a deployment can generally be upgraded with little to no downtime. When used in conjunction with a system like Amazon Web Services, the Unisubs Engine is a rapidly scalable and robust platform for community-driven captioning and translation of video.

Figure 4: Unisubs Readme Document



This repository is the code for the Amara project.

The full documentation can be found at http://amara.readthedocs.org/en/latest/index.html

Quick Start

Amara uses Docker. For ease of development, we use the docker-compose tool to have a full, production like, local dev environment.

1. Git clone the repository:

```
git clone git://github.com/pculture/unisubs.git unisubs
```

   Now the entire project will be in the unisubs directory.

2. Install docker-compose (http://docs.docker.com/compose/install/)
3. Build the Amara docker image:

```
bin/dev build
```

The steps for installing the Unisubs Engine can be found in the Unisubs Readme document[8]. For a seasoned developer or engineer, the process for getting the Unisubs Engine running is straightforward; what would have once required much more time and computational power is now quick, convenient, and operates on minimal resources. This conversion is a boon for any developer, researcher, or organization that is interested in running and/or improving the Unisubs Engine.

[8] https://github.com/pculture/unisubs/blob/production/README.markdown

3.2 Modularization: Subtitle & Team Workflows and Behaviors

3.2.1 INITIAL SITUATION

The Unisubs Engine began life as a prototype interface for captioning and subtitling web video. The prototype evolved into a more robust captioning/subtitling interface that won awards for advancements in accessibility and multi-culturalism. At this point in the platform's evolution, it was growing quickly and whenever the need for customization arose, modifications were made directly, deep inside the Unisubs Engine's core code. This led to a complicated tangle of customizations and special bits of code that were buried in the larger platform.

Modularity and the ability to easily and broadly customize the Unisubs Engine was not envisioned as one of the initial requirements. However, the need for this type of flexibility became increasingly clear as the platform grew in popularity. A system for customizing aspects of the platform would be especially powerful if it could manage to do so in clean, self-contained modules. This would unlock the ability for new and exciting approaches to community subtitling and translation to be explored and emerge more spontaneously.

3.2.2 GOALS & TARGETS

To build a more modular and reconfigurable framework for the Unisubs Engine, the product team drew on years of experience fostering community captioning and subtitling efforts. They approached the issue with a goal of building a generalizable framework that could act as a base, and be more readily expanded.

The product team began by designing a generic structure that would enable changes (large or small) to how the Unisubs Engine behaves. The result is called *Behaviors*, a framework that supports generic modularization. This concept allows a component inside the Unisubs Engine to define a “behavior function”, which other components could override to modify the behavior. It follows the Chain of responsibility pattern, described in Wikipedia as:

> A design pattern consisting of a source of command objects and a series of processing objects. Each processing object contains logic that defines

the types of command objects that it can handle; the rest are passed to the next processing object in the chain.[9]

Behaviors enables broad flexibility, allowing almost any aspect of the Unisubs Engine to be customized, while still maintaining a clean and modular code structure. Two new modules built on top of Behaviors include, *subtitle-workflows* and *team-workflows*, which both allow more specific types of customization in more clearly defined areas of the platform.

The team-workflows and subtitles-workflows modules enable customization of rules that govern how people find captioning/subtitling projects and work on them together. The specific customizations and rules are encoded in these modules. For instance, one team-workflow module might dictate that two people need to both give a 'thumbs up' in order for a set of captions to go live. Once this requirement is met the interaction model could change to an open Wikipedia-style collaboration, which would allow for additional community-created translations to be created by volunteer community members in many different languages. A different module might include a special dashboard for 'mentors' to see who is working on which video (and help them out). By leveraging these modular frameworks, workflows can be totally customized without having to modify any of the Unisubs Engine's core code.

With these building blocks in place, any researcher, developer, or organization could develop new modules that were independent, but still interoperable, with the Unisubs Engine. Of course any of these additional modules could be improved or collaborated on by others as well. The team's approach to modularity fits well within the free and open source software paradigm.

3.2.3 IMPLEMENTATION

The structure described above has been implemented in the Unisubs Engine and is documented in three sections:

- **Behaviors-framework** http://amara.readthedocs.io/en/apidocs/dev-guide/behaviors.html

[9] https://en.wikipedia.org/wiki/Chain-of-responsibility_pattern

- **Subtitle-workflows** http://amara.readthedocs.io/en/apidocs/dev-guide/subtitle-workflows.html
- **Team-workflows** http://amara.readthedocs.io/en/apidocs/dev-guide/team-workflows.html

Each of these key structures is now modular, allowing clean modification and customization.

3.3 Documentation: Beautiful and Systematic

3.3.1 INITIAL SITUATION

The Unisubs Engine sprung out of a prototype project and, while it has always been documented in some capacity, the documentation has not always been consistent, thorough, and easy to navigate. In fact, in earlier days the documentation was scattered and somewhat lacking in both presentation and usability. As the project evolved, the documentation also improved, but consistency, ease of use, and clear formatting still had significant room for refinement.

3.3.2 GOALS & TARGETS

The product team had two goals for improving the Unisubs Engine documentation: 1) Follow a style guide for in-code documentation consistency, 2) Integrate and automate the public presentation of documentation and ensure it's easy to follow.

To systemize in-code documentation, the product team decided to follow best practices set out in the Google Python Style Guide. These guidelines for commenting (and code documentation) are located at: http://google.github.io/styleguide/pyguide.html?showone=Comments#Comments A clean example of how this approach looks in practice, can be viewed here: https://sphinxcontrib-napoleon.readthedocs.io/en/latest/ Conforming to these standards creates in-code

documenation that is consistent, legible, and can be easily applied by any engineer or developer who is contributing code to the project.

To automate the presentation of the Unisubs Engine's documentation, the product team uses Read the Docs (RtD), a platform and host for documentation. RtD processes commented source code from the Unisubs Engine and transforms it into an easy to navigate and searchable web page. In their own words, "We [at RtD] build documentation and host it for you. Think of it as Continuous Documentation."[10] A primary benefit, when using a system like RtD, is that documentation is done once (in the code), and then automatically flows into an easy to read format. This is far superior to a process that might require in-code documentation that is then followed by creating and maintaining a more formalized public-facing documentation.

As an additional measure to make the platform and code easier to understand, the product team decided to implement browsable API endpoints. The REST framework provides an entirely self-describing API; in other words, each API endpoint has baked-in documentation that can be viewed with a browser.[11] This enables real-time interaction with the API through a web browser, allowing a person to more easily experiment, debug, and interact with the Unisubs Engine's API.

3.3.3 IMPLEMENTATION

The Unisubs Engine's most basic documentation is in the README doc on the public code repository: https://github.com/pculture/unisubs/blob/production/README.markdown. This document describes how to get the Unisubs Engine up and running in an environment using Docker's docker-compose tool.

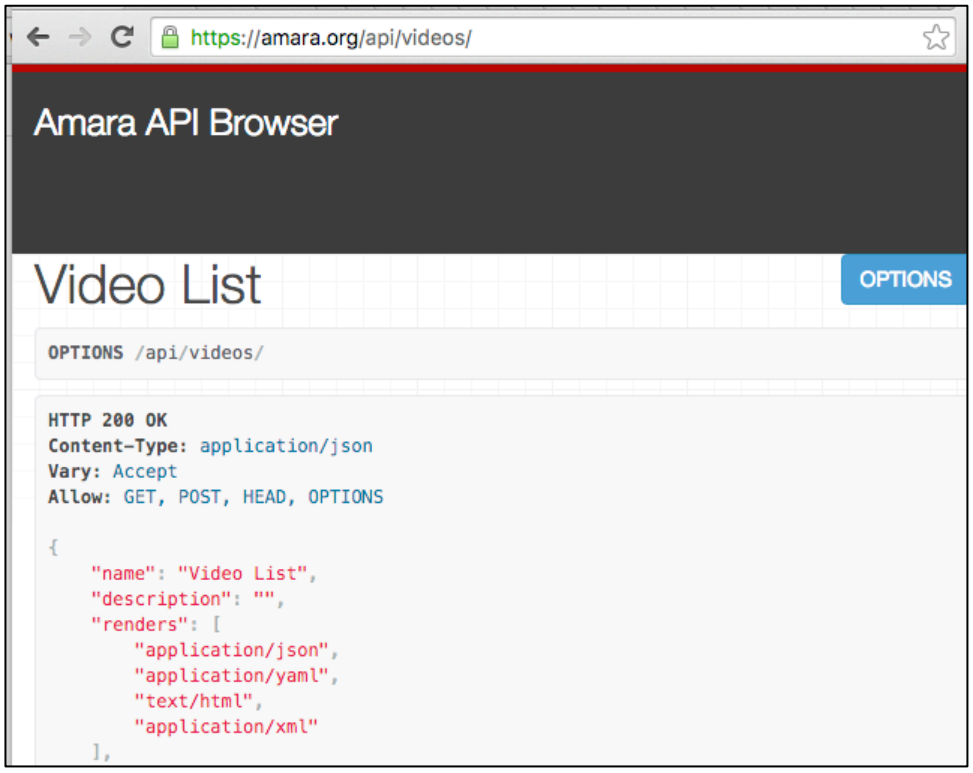
Broader documentation for the platform, API, and modular components are available through Read the Docs: http://amara.readthedocs.io/en/latest/api.html (Note: These documents are hosted under the Amara name, due to broader recognition.) Selected sections of the documentation, mentioned in this report, include:

- **API Documentation** http://amara.readthedocs.io/en/latest/api.html

[10] https://docs.readthedocs.io/en/latest/index.html#user-docs
[11] http://www.django-rest-framework.org/topics/documenting-your-api/#self-describing-apis

- **Modular Components**
    - Subtitle Workflows – http://amara.readthedocs.io/en/apidocs/dev-guide/subtitle-workflows.html
    - Team Workflows – http://amara.readthedocs.io/en/apidocs/dev-guide/team-workflows.html
    - Behaviors – http://amara.readthedocs.io/en/apidocs/dev-guide/behaviors.html

Figure 5: Unisubs Browsable API Endpoints



Additionally, the browsable API endpoints are viewable at any installed version of the Unisubs Engine. The endpoints of Amara's production server can be viewed here: https://amara.org/api/ (Note: Using the Amara endpoint browser requires the viewer to be logged into the service).

4 Conclusion

The work done to-date to improve the Unisubs Engine's documentation, modularity, and infrastructure provides a solid base for community development; however, the work presented in this report is by no means a finished product. As developers and engineers expand and build on the Unisubs Engine, the platform, documentation, and infrastructure will continue to be used, tested, and refined. It is this iterative process that goes to the heart of the Unisubs Engine –and Amara's– vision for a more engaging and collaborative world that allow broader participation.

Annex I: Glossary

Abbreviation	Full form
AAL	Ambient Assisted Living
ACS	AsteRICS Configuration Suite
AoD	Assistance on Demand
API	Application Program Interface
AsteRICS	Assistive Technology Rapid Integration & Construction Set
AT	Assistive Technology
C4A	Cloud4All
D	Deliverable
DoW	Description of Work
DSpace	DeveloperSpace
GUI	Graphical User Interface
GPII	Global Public Inclusive Infrastructure
ICT	Information and Communications Technology
IDE	Integrated Development Environment
ISO	International Organization for Standardization
IT	Information Technology
KPI	Key Performance Indicator
P4A	Prosperity4all
PCF	Participatory Culture Foundation
R&D	Research and Development
RtD	Read the Docs
REST	Representational State Transfer
SP	Sub-Project
UI	User Interface
UX	User Experience
VM	Virtual Machine

Abbreviation	Full form
WP	Work Package