D2.2 - Möbius Technology Blueprint

Project ref. no.	957185
Project title	Möbius: The power of prosumers in publishing
Project duration	1st March 2021 – 28th February 2024 (36 months)
Website	www.möbius-project.eu
Related WP/Task	WP2 / T2.3
Dissemination level	PUBLIC
Document due date	30/11/2021 (M9)
Actual delivery date	06/12/2021 (M10)
Deliverable leader	IN2
Document status	Final

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 957185

Revision History

Version	Date	Author	Document history/approvals
0.1	27/04/2021	George Ioannidis (IN2)	First draft TOC and content
0.2	21/05/2021	Niklas Reppel (EUT)	Integration of input regarding EPUB tools, 3D and binaural audio production
0.3	01/06/2021	Natascha Pattock (MVB)	Integration of input regarding metadata standards
0.4	01/06/2021	Gino Querini (FEP), Enrico Turin (FEP)	Integration of input regarding the publishing workflow
0.5	10/09/2021	Thomaso Greco (Bookabook)	Integration of input regarding book crowdfunding
0.6	17/11/2021	Alexandru Stan (IN2)	Elaboration of functional requirements
0.7	24/11/2021	Alexandru Stan (IN2)	System architecture
0.8	26/11/2021	Alexandru Stan (IN2)	API definitions
0.9	29/11/2011	George Ioannidis (IN2)	Release for internal review
0.9.1	29/11/2021	Simona De Rosa (DEN)	Review
0.9.2	29/11/2021	Elena Brunello (Bookabook)	Review
1.0	30/11/2021	George Ioannidis (IN2)	Integrated review comments, fixed tables and figures and released for submission.
1.1	06/12/2021	Patricia Castillo (EUT)	Final revision and formatting before submission.

Executive Summary

This deliverable set-up the foundations for the technical developments of the Möbius tools. It gathered the different background technologies, tools, libraries and standards that are relevant for the developments. The expected technology outcomes of the project were afterwards presented: the Prosumer Intelligence Toolkit, the Möbius Creator's Toolkit and the Möbius Player. The user requirements gathered were then summarised and analysed. Requirements that were out of scope were identified and for all the others it was the implementation implications were distilled based on three main aspects: backend, frontend and storage. Based on this, for each of the Möbius tools the functional and system requirements were specified. In total 44 requirements were identified for the backend components, 40 requirements were identified for the user interfaces and 28 requirements were identified for the storage. All these requirements were prioritised. Additionally, 19 further requirements were extracted for the Möbius container format. Architectural diagrams for each of the Möbius envisioned outcomes were presented as well as API descriptions for the data models and related methods. Finally, the approach for user feedback and integration process was presented.

This deliverable represents the main outcome of Task 2.3 "Architecture, logic and interfaces" and it contains the guidelines to assist the design of software integrations and tools, including prioritized requirements based on the user feedback. It also contains the outcome of surveyed tools and technologies relevant for Möbius which was carried out in T2.1 "User-driven innovation framework for enriched publishing".

Table of Contents

List of Figures

List of Tables

Terminology and Acronyms

Bx	*Backend system requirement #x*
BDS	*Bibliographic Data Service*
BIC	*Book Industry Communication*
BRIR	*Body-Related Impulse Responses*
CMS	*Content management system*
DAW	*Digital audio workstations*
EC	*European Commission*
EU	*European Union*
FP	*Framework Programme*
Fx	*Frontend system requirement #x*
FRx	*Format requirement #x*
GTIN-13	*Global Trade Identification Number*
HRTF	*Head-Related Transfer Function*
KDP	*Kindle Direct Publishing*
MVC	*Model-View-Controller*
PIT	*Prosumer Intelligence Toolkit*
PMB	*Project Management Board*
PMP	*Project Management Plan*
Rx	*Requirement #x*
RC	*ranked by the consortium partners*
RE-U	*ranked by end-users*
Sx	*Storage system requirement #x*
SEA	*Search Engine Advertising*
SEO	*Search Engine Optimisation*
STAB	*Scientific and Technical Advisory Board*
TTS	*Text-to-Speech*
WP	*Work Package*

1 Introduction

This deliverable represents the main outcome of Task 2.3 “Architecture, logic and interfaces” and it contains the guidelines to assist the design of software integrations and tools, including prioritized requirements based on the user feedback. It also contains the outcome of surveyed tools and technologies relevant for Möbius which was carried out in T2.1 “User-driven innovation framework for enriched publishing”.

The deliverable builds on the user requirements presented in D2.1 and its main aim is to operationalize the data and tools that will make possible meaningful cooperation with the users. Lead by IN2, in close cooperation with EUT, this work defines an organised set of functional and system requirements and the high-level architecture and workflow of the software tools and the digital assets to be integrated and developed as part of the Möbius technological solutions:

- the computational social science methods for better prosumer intelligence and business models,
- and the Möbius book.

The document is organised as follows:

- Chapter 2 describes the background technologies that are relevant as well as provides additional information on the specificities of the publishing houses and crowdsourced publishing
- Chapter 3 describes the envisioned technology outcomes of the project, giving an overview of the different tools that will be developed
- Chapter 4 summarises the user requirements that have been gathered
- Chapter 5 presents the functional and system requirements that have resulted from the user requirements
- Chapter 6 describes the architecture and user workflows of the Möbius tools
- Chapter 7 informs about the process for integrating user feedback
- Chapter 8 presents the conclusions of this report

2 Background

In order to understand better the context and the background of the Möbius project in relation to its envisioned outcomes we carried a desk research exercise to identify the most relevant tools, processes and workflows for Möbius. In this desk research we looked both externally (third-party tools) and also the main contributions that the consortium partners can bring, and have identified the following main categories:

- EPUB tools and related products
- Tools for 3D audio production

- Tools for discourse and controversy identification and exploration
- Tools for media management and publishing
- Tools for web-based storytelling
- Workflows and stands in publishing houses

The next sections give an overview of this desk research.

2.1 Third-Party EPUB Tools & Related Products

The following paragraph provides an overview about already existing tools that allow editing multimedia-enabled e-Books and other related products.

2.1.1 Relevant Standards

According to our analysis, there are several standards that are relevant in this context:

- MPEG-4 Part 14 (mp4) file format[1]: relevant for audiobooks with bookmarks, chapter markers, etc
- EPUB 3.2[2]: the main W3C open standard format for the distribution and interchange of digital publications and documents. EPUB3 represents the key format to be used and extended for the Möbius Book.
- EPUB 3.3[3]: the current W3C working draft for the next EPUB3 specification.
- EPUB Media Overlays[4]: W3C specification part of EPUB3.2 dedicated to the specification of text and audio synchronization enabled by Media Overlays. This will be a key functionality for the Möbius Book.

2.1.2 EPUB3 Authoring Tools

Looking at the existing tools that allow creating eBooks in EPUB format, with more or less multimedia capabilities, it is possible to mention the following ones:

- Tobi[5] – Tobi is a free, open-source, multimedia book production authoring tool for creating human narrated talking books synchronized with text and images conforming to the DAISY3[6] and EPUB3 standards.
- Sigil[7] – Sigil is an open-source application designed to make it easy to create eBooks using the EPUB format. It can be used to format and package books into an EPUB. From a first review, it seems that it has limited multimedia capabilities.

[1]https://en.wikipedia.org/wiki/MPEG-4_Part_14
[2]EPUB 3.2, https://www.w3.org/publishing/epub3/index.html
[3]EPUB 3.3, https://www.w3.org/TR/epub-33/
[4]EPUB Media Overlays, https://www.w3.org/publishing/epub32/epub-mediaoverlays.html
[5]Tobi, https://daisy.org/activities/software/tobi/
[6]The current DAISY Standard (DAISY 3) is officially the ANSI/NISO Z39.86-2005 (R2012) Specifications for the Digital Talking Book. https://daisy.org/activities/standards/daisy/daisy-3/
[7]Sigil, https://sigil-ebook.com/sigil/

- ViewPorter[8] - ViewPorter is a freemium application for creating eBooks and provide a set of tools that include cloud storage and PDF to EPUB3 conversion.
- EPUBeditor[9] – EPUBeditor is a freemium online environment for creating eBooks which also can contain multimedia content and interactive features.
- Kotobee[10] – is a commercial platform for creating interactive eBooks

2.1.3 Reading Apps & Related Products

With regards to apps used for reading eBooks, we have identified:

- Amazon/Audible immersion reading (aka WhisperSync), allowing to seamlessly switch between reading and listening[11]
- Menestrello[12] – Free alternative to Amazon products, reads files created by the Aeneas library (see also below).
- Colibrio reader[13] - is a closed sourced SDK for developing accessible reading experiences implementing from scratch the EPUB3 standard

2.1.4 Related Libraries

The most commonly used libraries and software development kits to create EPUB related applications are:

- Aeneas[14] - a Python/C library and a set of tools to automatically synchronize audio and text (aka forced alignment).
- Epub-creator[15] - a Java library to build valid EPUB3 formats
- Epublib[16] - a Java library for reading/writing/manipulating EPUB files
- Readium[17] - a playground for various languages (includes a C++ renderer) for digital reading technologies
- Epub.js[18] - a JavaScript library for rendering EPUB documents in the browser, across many devices
- Bibi[19] - a JavaScript EPUB reader for websites

[8]ViewPorter, https://viewporter.com/
[9]EPUBEditor, https://www.epubeditor.it/home/info-en/
[10]Kotobee, https://www.kotobee.com/
[11]What is Immersion Reading? https://help.audible.com/s/article/what-is-immersion-reading?language=en_US
[12]Menestrello, https://www.readbeyond.it/menestrello/
[13]Colibrio Reader, https://colibrio.com
[14]Aeneas library, https://www.readbeyond.it/aeneas/
[15]epub-creator, https://github.com/OpenCollabZA/epub-creator
[16]epublib, https://github.com/psiegman/epublib
[17]Readium, https://github.com/readium
[18]Epub.js, https://github.com/futurepress/epub.js
[19]Bibi, https://github.com/satorumurmur/bibi

- DeepZen[20] - a premium API-based tool for text-to-speech conversion

2.2 Tools for 3D and binaural audio production

Eurecat has a suite of 3D audio production tools that integrate into the typical workflow of digital audio workstations (DAWs). A more in-depth description of these tools is available in D4.1 “Möbius core technologies".

2.2.1 Sfëar 3D Audio Suite

A suite of plugins for digital audio workstations like Logic Audio or Ableton Live (AU plugin format) developed by EUT, that, with the help of an external server, is designed to produce spatial audio content in the Ambisonics format. The Ambisonics format allows for great flexibility, as the content can be rendered and reproduced on a variety of speaker configurations as well as on headphones, in binaural stereo, using Eurecat’s proprietary decoder formats. With the help of head-tracking, that is, the tracking of the user’s head movements, the user can “move around’ in the sound scene, leading to an elevated sense of immersion. Thanks to the client-server architecture, the Sfëar suite enables users to produce Ambisonics content on platforms (like Ableton Live) that would otherwise only allow for stereo productions. The included visualizer allows to position the sound sources in an intuitive manner. Content produced with this product can be played back with the Sfëar Player (see below) or any other player that can play back Ambisonics audio. Additionally, the product can be used to export productions in static formats, either binaural stereo or the speaker channels for a specific speaker configuration.

2.2.2 Eurecat SOFA Panner

The SOFA panner is a plugin for digital audio workstations like Logic Audio or Ableton Live (VST3/AU/AAX formats) that enables the positioning of audio sources in static binaural productions. The sound sources in this plugin are spatialized directly by convolving the audio stream with a Head-Related Transfer Function (HRTF), that is, by applying digital filter that mimics the filter characteristics of the human head and ear, thus re-creating the impression of spatial hearing over headphones. The process is very similar to a convolution reverb, except that by adding characteristics of a room, we are adding characteristics of the human ear depending on the direction of arrival of the sound. In fact, as an alternative to HRTFs, so-called Body-Related Impulse Responses (BRIR), which include acoustic information about the room they were recorded in, can be used to spatialize sounds with a certain amount of room reverb. As the name suggests, the HRTF of BRIR filters are stored in the widely available SOFA format, which is specifically designed to hold and lookup the position-dependent filters[21]. Using the various SOFA files (files that contain HRTF or BRIR data) that are either publicly available or in the hands of Eurecat, the SOFA panner thus allows to create a wide range of spatial (room) impressions. As the export format in this case is static binaural stereo, it can be played back by any music player application.

[20]DeepZen, https://deepzen.io/
[21]see https://www.sofaconventions.org/ for more information about SOFA

2.2.3 Sfëar 3D Audio Player

The Sfëar Player is a 3D audio/music player application (for macOS and iOS, potentially other platforms). It allows a user to reproduce audio content in various formats, including Ambisonics up to fifth order. Same as in the production suite, the Sfëar player allows to reproduce the content on speaker setups as well as on headphones in binaural stereo. The iOS version also supports using the AirPod Pro's integrated gyroscope to perform head tracking when playing back Ambisonics content, thus leading to an enhanced feeling of immersion. With small modifications, this product could also be used to reproduce immersive audiobooks.

2.2.4 3D Audio Libraries

The 3D audio libraries represent the foundation of EUT's spatialization technology and have been used to develop all the above-mentioned products. Being a suite of product-independent C++ libraries, all of Eurecat's spatialization technology can be integrated in a wide range of solutions with their help, potentially even web-based solutions if cross-compiled to WebAssembly. The libraries include:

- CatConv – a fast convolution library, i.e., for spatializing with the help of HRTFs
- CatSonics – Ambisonics audio processing and rendering
- CatArt – scene-based spatialization on top of CatSonics

Also included is a set of prototype plugins that exist as test cases for the respective libraries. These prototypes can be used to create and manipulate Ambisonics audio material in DAWs that allow multichannel tracks, such as REAPER[22] or Ardour.[23] They aren't meant for publication as they are but could serve as a component in the overall product with some modifications.

2.3 Discourse and controversy identification and exploration

Previous work related to the Prosumer Intelligence Toolkit from EUT are contributions to the developments of Contropedia, an interactive platform for exploring controversies in articles from MediaWiki powered platforms.[24]

Moreover, EUT has carried out the first large-scale quantitative and qualitative study of the digital social reading platform Wattpad[25] as well as research on literary modelling and statistical analysis to study a variety of digital social reading platforms (Goodreads, Archive of Our Own,

[22] REAPER Digital Audio Workstation, https://www.reaper.fm/

[23] Ardour – Record, Edit and Mix on Lnux, MacOS and Windows, https://ardour.org

[24]Borra, E., Weltevrede, E., Ciuccarelli, P., Kaltenbrunner, A., Laniado, D., Magni, G., Mauri, M., Rogers, R., & Venturini, T. (2014). Contropedia - the analysis and visualization of controversies in Wikipedia articles. In Proceedings of The International Symposium on Open Collaboration. OpenSym '14: The International Symposium on Open Collaboration. ACM. https://doi.org/10.1145/2641580.2641622

[25]Pianzola, Federico, Simone Rebora, and Gerhard Lauer. 2020. "Wattpad as a resource for literary studies. Quantitative and qualitative examples of the importance of digital social reading and readers' comments in the margins". PLoS ONE 15.1: e0226708. Data and code: https://osf.io/5gxmn/

Wattpad)[26]. The study focuses on the evolution of reading technologies, intervening in the debate about the impact of digital technology on reading, discussing the pedagogical benefits of digital social reading, and organizing the knowledge produced in 15 years of research about digital social reading into useful taxonomies.

2.4 Media Management and Publishing

MyMeedia[27] is a flexible media management and publishing platform developed by IN2. It comprises of 4 main building blocks:

- Importing and Ingesting: imports everything from multimedia files, documents and photos, to social streams, web feeds, third-party APIs and open data.
- Processing: includes advanced AI content indexing and analysis components to crunch all content and make sense of your data, extract additional information and enable powerful search and retrieval.
- Enriching: provides easy to use tools to geotag, annotate and enhance content with additional keywords, tags and metadata that are relevant
- Accessing: provides an API so that developers can easily build rich user experiences and interactive applications without worrying about the underlying content infrastructure

It includes a web-based authoring environment for the creation of mixed-media stories, a web-based application for the creation and management of content hubs and a series of dashboards for the visualisation of data and content metadata. The platform also contains modules for content analysis, enrichment and automatic annotation, content aggregation, indexing, syndication and publishing in 3rd party websites or applications.

A more in-depth description of MyMeedia is available in D4.1 “Möbius core technologies”.

2.5 Web-based storytelling

Tellit - Content Management and Storytelling for the Social World[28] is a web-based application for managing and re-using social media and web content alongside audio-visual content. Tellit combines aggregation of third-party content (e.g., from social media, RSS feeds, external APIs) with an intelligent and semi-automatic way to create data-driven clusters and topics. The tool also provides a range of web-based adaptive and responsive layouts to present a story. A more in-depth description of these tools is available in D4.1 “Möbius core technologies”.

2.6 Workflows and standards used by a publishing house

A publishing house is made of several departments performing different functions. Depending on the nature, scale, and scope of the publishing house these can vary in terms of internal

[26]Pianzola, Federico. 2021. Digital Social Reading: Sharing Fiction in the 21st Century. (MIT Press open peer review).
[27]MyMeedia - Multimedia Content Management and Publishing, https://mymeedia.com
[28]Tellit - Content Management and Storytelling for the Social World, https://tellitapp.com/about

relationships and relative size. As with most of the sector, the digital transformation is currently transforming pre-established roles (e.g., the limits between sales and rights departments). A rough non-exhaustive taxonomy of these functions/departments might look as follows:

1. **Administration/management:**

Under this heading we can list the publisher as owner/ leader of the publishing house. This does not necessarily mean that the publisher is the administrator of the publishing house, a separate role.

The commissioning editor is the one in charge of the publishing house identity, through the publishing list and overview on the editing.

The contract department oversees the contracts with the authors – not necessarily in regard to rights, which can be handled by a dedicated rights department.

2. **Editorial department:**

Under the commissioning editor direction, the editorial production oversees editing activities in collaboration with the author and the design department. The editors' role is tied most closely to the texts' contents and structure. Editors might be in-house or freelancers.

Editorial content might also be related to translation (again, in-house or freelance). Editing also relies on professional readers' evaluations on manuscripts. Proof-readers' activities also belong to editing and they as well can be in-house or freelance.

3. **Design department:**

In this department both design of the book cover and pagination take place. This work is developed in collaboration with the editorial department and the production department. As for other departments, designers might be freelance or in-house. Designers also collaborate with the production department to ensure the production of the final product is viable.

4. **Production department:**

This department is in charge of dealing with all the activities that will bring the edited text to physical (in case of print) realization. This means handling provisions of raw materials, printing, and binding. This means that this department works in close collaboration with administration and design to mediate budgets, designs and actual feasibility.

In case of completely electronic production the digital department is in charge of the file preparation, conversion in adequate format and digital storage.

5. **Marketing department:**

This department oversees the promotion of books to retailers and wholesalers in cooperation with the sales department. Their communication activities rely both on numerical data in terms of sales and audiences as well as qualitative ones (e.g., how a title fits in a trend).

6. **Sales department:**

The sales department organizes the sales strategy, negotiating with customers (retailers, distributors, etc.).

2.6.1 The publishing workflow

Publishing workflows differ greatly depending on countries, publishers, even outputs[29]. Similarly, they vary depending on the desired output and the nature of the publishing (self-publishing, publishing via a publishing house, publishing in international partnership, etc). Furthermore, the digital transformation affects the workflow and adds to its complexity. Key differences in workflows depend on them being or not being **digital-first**. A traditional publishing house workflow roughly follows these phases.

1. Editorial commission

In this phase, the commissioning editor, either proactively or reactively, looks for new titles for the publishing house. This means either commissioning an author, evaluating received proposals or utilizing any other form of acquisition strategy. The result of the editor's activity is the **publisher's list**, which defines the identity of the house/imprint simply by its selection of titles.

2. Manuscript acquisition (IP acquisition)

In terms of metadata handling this phase is relevant as the signature of the agreement between publishing house and author acts as trigger for the ISBN allocation to the title; in terms of difficulties with the ISBN as metadata there are the following issues:

- Self-publishing, especially on Amazon's Kindle Direct Publishing (KDP) do not rely on ISBN. **For those titles** Amazon might rely solely on its ASIN code.
- Other self-publishing authors might not use ISBN at all.

3. Editing phase (content editing, copy-editing)

In the copy-editing phase the text is circulated among editors and authors for the revision of content and structural integrity. If needed, a text can be formatted following an internal or standard style guide (e.g., Chicago Manual of Style). Parallel to this process the text undergoes (if not originally conceived as such) **structural markup.** This markup can indicate elements such as chapter headings, sections, sub-section headings, etc. as well as other elements such as long quotations, lists, notes, captions, tables, etc. to serve in the design phase (the typographical markup which will result in the page-proofs and final product before print).

4. . Design and production (cover, typesetting)

In this phase the text is handled by a typesetter, which following the indications from the editing proceeds to the **typographical markup** of the text. This means that the indications from the editing are transformed into an **xml** markup which in turn will generate the final text in pdf format. The xml file is the one that will be archived and allow the publisher to re-purpose the original content for different outputs (e.g., having marked up something as "chapter title" will

[29] Best practices for Publisher Workflows, https://www.accessiblepublishing.ca/accessible-publishing-best-practices-for-publisher-workflows/

then allow for page proofs for physical printing or the generation of a clickable Table of Contents for eBooks). This process undergoes several revisions before finalization, too. This phase also includes the design of the cover of the work (which starts way before the text is finalized) and is one of the elements through which the text is initially marketed and presented to the audiences (be that other businesses or readers).

5. Marketing and distribution

- Storage
- Order processing
- Distribution

Marketing goes parallel to issues of storage and distribution. Selling the book implies managing orders and logistics. In a pre-digital workflow this meant simply that the typeset text was printed, bound, and then stored/distributed. In a digital workflow this means several different things:

- If the text is conceived originally as digital:
    - "Storage" means digital storage- with different (cheaper) requirements than physical storage (CMS: content management systems)
    - Digitally stored texts are always accessible, thus affecting the notion of a backlist.
    - Digital text, depending on the granularity of their markup might be collated in different products (e.g., anthologies, selections, etc)
    - The use of Print-on-Demand technologies can remove the requirement for physical storage for printed works as well.
- If the text is not digital-first or if it has been produced in pre-digital workflows:
    - The markup might come after the typeset pdfs are produced; this slows down the production, might cause delays in the production (at least of the digital goods) and might produce inconsistencies.
    - The text might need digitalization from old physical copies. This might mean rekeying or OCR scans than only then are marked up and properly digitalized

Regarding **marketing** and **distribution overall** it is necessary to point out several things

- The metadata associated with a work are key in finding the work, distributing it. Usually these include (but are of course not limited to):
    - ISBN, book title, author, cover, description
- Depending on the granularity and precision of these data the product might be easier to find/sell:
    - This includes activities such as listing the text- via its metadata- in different bibliographical repositories/aggregators.

6. Sales

Publishing houses sales usually are not B2C but from publishers to retailers, distributors, etc. Direct sales are however the norm for scholarly publishers (with research institutions and libraries being the main customers) and can be a significant component of sales for educational publishers too when public authorities purchase the materials.

An important aspect of this phase, relative to Möbius, is the publishers' engagement with user behaviours and the function of metadata in tandem with marketing activities, tied with the use of digital formats such as EPUB3. Regarding user behaviours, with EPUB3, publishers can both embed audio/video content in their eBooks as well as get greater user interaction via JavaScript. Publishers can also gain greater insights in user behaviours via JavaScript as well.

Regarding metadata, EPUB3 allows to define the metadata scheme (e.g., MARC, ONIX) and allows for a much richer set of metadata than EPUB2. Furthermore, EPUB3 allows to associate metadata at component and element level, not only file, generating what is known as **rich metadata** (semantically richer metadata useful for algorithm-ruled placements)**.**

2.6.2 eBook Distribution

To 'distribute' an eBook is to provide it to online retailers for sale (and therefore download). This can be done either directly by using a self-service online retailer or through an aggregator. Some companies, like Kobo, are both retailers and aggregators. Before distributing an eBook, the eBook file has to be formatted properly and a cover has to be designed.

1. On Formats

There are two main formats that are used by eBook distributors, retailers, and e-reading devices: EPUB and .mobi. The main difference between them is simple: .mobi is Amazon's proprietary format (and only used by them), while EPUB is used by everyone else. There are conversion tools available online which transform a manuscript into an EPUB file.

In some countries an ISBN is a precondition before distributing an e-book. The advantage of buying an ISBN for the e-book is that the author himself is the publisher and not the Online-platform which makes the e-book available. For each format of the book an ISBN is needed.

2. The cover design

For the cover design it is important which requirements the retailer has on the format (number of pixels, jpg, gif...) On top of your EPUB or .mobi, the other file you'll need to upload is your book cover file.

3. The decision on retailers

Amazon controls around 80% of the eBook market in the US and UK, but their market share is much lower in other countries. In Canada, for example, Kobo alone controls over 25% of the e-book market. In Germany, Tolino has the same market share as Amazon. There is a wide range of major retailers for the distribution of eBooks.

- Amazon Kindle

- Tolino
- Apple Books
- Kobo
- Barnes & Noble
- Google Play

eBooks can be sold directly by retailers or “indirect” by using an aggregator. When going directly to retailers, the eBook file and the metadata must be uploaded by the author himself. If there are changes, the eBook file has to be uploaded again to each individual platform. The sales reports are also provided by each platform itself, so that there might be a huge amount of sales information coming from each platform at different times. On the other hand, by using an aggregator[30] the eBook can be distributed to a large number of stores at once. The aggregator consolidates the metadata and sales reports so that the author has one source for the distribution and information process.

2.6.3 Metadata Standards

In terms of their distribution publishers usually provide books metadata to bibliographers’ databases such as **Nielsen BookData**[31] or “**Verzeichnis lieferbarer Bücher**[32]**” – the German Books-in-Print Database.** These databases offer different formats for publishers depending on the level required by the customer: from basic bibliographic information to more sophisticated information. Other suppliers of metadata such as e.g., **Bibliographic Data Service (BDS)** specialize in working with libraries (utilizing the MARC format[33]).

It is important to not confuse descriptive standards (such as BISAC[34]), with the metadata technical standard associated with a file and/or a format. E.g., EPUB3 utilizes the **Dublin Core Metadata Element Set**[35] which lists 13 possible properties for any document (within which one is reserved for “subject” and the others refer to the document identifier and details).

[30]Example of aggregators include: Draft2Digital (https://draft2digital.com/), Smashwords (https://www.smashwords.com/), XinXii (https://www.xinxii.com/), Publish Drive (https://publishdrive.com/), Kobo (https://www.kobo.com/), Libreka (https://info.libreka.de/en/)

[31]Nielsen BookData Online: https://www.nielsenbookdataonline.com/bdol/

[32]Verzeichnis lieferbarer Bücher, https://vlb.de/

[33]MARC (machine-readable cataloguing) standards are a set of digital formats for the description of items catalogued by libraries, such as books. https://www.loc.gov/marc/

[34]BISAC (Book Industry Standards and Communications) is a subject vocabulary for books created by the publishing industry, specifically the Book Industry Study Group (BISG). It is arranged hierarchically and includes codes as well as textual labels for entries. BISAC is commonly used in bookstores, and has been seen in action in Google Book Search. https://en.wikipedia.org/wiki/Book_Industry_Study_Group

[35]Dublin Core™ Metadata Element Set, Version 1.1: Reference Description https://www.dublincore.org/specifications/dublin-core/dces/

In the publishing industry “**ONIX for Books”**[36] has been established by **EDItEUR** as a standard for book metadata. EDItEUR is an international group coordinating the development of standards infrastructure for electronic commerce in the book, e-book and serials sectors. The ONIX for Books Product Information Format is the international standard for representing and communicating book industry product information in electronic form. It is an **XML-based standard** for rich book metadata, providing a consistent way for publishers, retailers and their supply chain partners to communicate a wide range of information about their products. It is overtly a commercial data format, is expressly designed to be used globally, and is not limited to any language or the characteristics of a specific national book trade. It’s widely used throughout the book and e-book supply chain in North America, Europe and Australasia, and is increasingly being adopted in the Asia Pacific region and South America too.

As an XML-based standard, each release of ONIX for Books consists of documentation that specifies the data content of a standard ONIX data file or ‘message’, plus an associated XML schema that can be used for validation of an ONIX file.

ONIX is not in itself a database, nor is it even a design for a database – it is a way of communicating data between databases.

For publishers who implement ONIX using an in-house development or by installing a commercial off-the-shelf product data management solution, experience has shown that ONIX for Books can be used as a communication format that makes it possible to deliver rich product information into the supply chain in a standard, consistent form, to wholesalers and distributors, to larger retailers, to data aggregators, and to affiliate companies.

It reduces the need to deal with multiple proprietary data formats, and hence reduces support costs. And by enabling third parties such as trade associations of data aggregators to develop metrics for data quality and timeliness, it encourages the overall improvement of the data available throughout the supply chain.

The backbone of standardization in the supply chain is the **ISBN**. It is the most successful product identifier ever devised and was introduced to replace individual publisher’s catalogue numbers and to enable the first drive to electronic commerce. Without clarity of identity, it was not possible to use computers to manage the supply chain. The entire structure of EDI (Electronic Data Interchange) standards on which an effective book supply chain has been built over the past 40 years has only been possible because of the implementation of the ISBN— distinguishing hardback from paperback, third edition from fourth edition.

A document’s metadata should at least reach the **Book Industry Communication** (BIC) standards, but the future of metadata lies in **semantic** metadata rather than data dedicated solely to the identification of documents. Data on books might deal with the documents’ themes, content, topics, etc. and can also go as far as touching reading context and expectations (e.g., summer light reading, etc.). The degree of penetration of metadata

[36]The ONIX for Books Product Information Format is an international standard for representing and communicating book industry product information in electronic form. https://www.editeur.org/83/Overview/

regarding a product is referred to as **granularity.** **The** more sophisticated the description of a document, of its parts, the easier for it to be found and circulated, both as a business level as well as a customer level.

In terms of semantic metadata their possible structures are classified as follows:

- **Controlled vocabularies**: an agreed upon and standard list of terms for metadata tagging.
- **Taxonomies**: a taxonomy of subjects – these are subjects listed in hierarchical fashion. E.g.: architecture -> religious -> churches
- **Thesauri**: forms of taxonomies with horizontal connections as well (between terms) indicating not only classifications but also relations between elements. The relationship might be of synonym or not. For example, we might correlate “churches” with “cathedrals”, with “monasteries”, “religious” with “cult”, etc.
- **Ontologies:** even more sophisticated than thesauri, they add what is called **faceted specification** to the properties listed in the thesauri. That does not mean introducing new forms of subject-properties relationships but more sophisticated descriptions for terms. Differently than thesauri, ontologies are open and extendable at will. For example, in the case of “church” we might specify the buildings’ locations, dates of construction, etc.

ONIX 3.0 - The message

An ONIX for Books message must have a header which identifies the sender, and optionally, the addressee(s), and carries a date stamp. The body of the message consists of an unlimited number of product records, each relating to a single product. Generally, this can be taken to mean a tradable product but there are exceptions. A product record can also be used to describe an item which is sold only as part of a set; or a set of items which are only sold separately; or a piece of promotional material which is offered to retailers, but which is not itself for sale or a trade pack intended to be broken up by a retailer for sale as individual items.

ONIX 3.0 - The product record

The product record begins with a few elements of record metadata among which a record identifier and a coded notification type are mandatory. A product identifier is also required, typically in the form of a GTIN-13 (Global Trade Identification Number) – usually an ISBN-13, unless the record refers to a non-book item whose GTIN comes from a different source. Alternative forms of product identifier can also be sent.

1 Descriptive detail (Bibliographic information)
2 Collateral detail (Marketing texts, links and objects)
3 Content detail (Content information)
4 Publishing detail (Publisher's information, status and selling rights)
5 Related material (Connection to other products)
6 Product supply (Information related to the supply chain)
7 Promotion detail (To describe promotional events intended to promote the product)

ONIX code lists

An important part of the ONIX for Books framework is the set of ‘code lists’, a series of controlled vocabularies that are used with particular XML data elements. Code lists can be revised without necessitating a new release of the main XML message specification and are the main means of ensuring ONIX can remain relevant as new business requirements arise.

Element	O30 List	O30-Reference Tag	O30 Short Tag
Product form	List 150	ProductForm	b012
Product form detail	List 175	ProductFormDetail	b333
Product form description		ProductFormDescription	b014
Product form feature	List 79	ProductFormFeature ProductFormFeatureType	Productformfeature b334
E-publication Accessibility Details	List 196		
Primary product content	List 81	PrimaryContentType	x416
Product content type	List 81	ProductContentType	b385
Type of DRM	List 144	EpubTechnicalProtection	x317
Usage constraints	List 145	EpubUsageConstraint	
Extent	List 23, 24	Extent ExtentType, ExtentValue ExtentUnit	b218 b219 b220
Sales rights	List 46	SalesRights SalesRightsType	Salesrights b089

Table 1. ONIX code lists

Product form

The product form indicates the primary form of a product and defines the generic category of a product. By specifying, for example, you determine whether a product is a book or an e-book. The form of a product must be specified in <ProductForm> or <b012>.

ONIX Code list for product form: **https://ns.editeur.org/onix/en/150**

Example ONIX-30 Short-Format	Example ONIX-30 Reference-Format
<b012>EA</b012> Digitalprodukt / E-Book	<ProductForm>EA</ProductForm>

Table 2. Product form

Product form detail

For a more detailed description of the product form or to provide added detail of the medium and / or format of the product, you can store detailed product form information in <ProductformDetail> or <b333>. For example, a book is a paperback, e-book format is EPUB, audio tracks are in MP3 format, etc.

In the detailed product forms, you will find a wide selection to describe your product in more detail according to the product form. In the code lists you will find information on which product form code may be combined with which detailed product forms. The information is repeatable.

ONIX Code list for product form detail: **https://ns.editeur.org/onix/en/175**

Example ONIX-30 Short-Format	Example ONIX-30 Reference-Format
<b333>E101</b333> EPUB <b333>A311</b333> Background music	<ProductFormDetail>E101</ProductFormDetail> <ProductFormDetail>A311</ProductFormDetail>

Table 3. Product form detail

Product Content

Product content indicates what types of content appear in a product. Which are closely related to but not strictly an attribute of product form. For example, you can specify that the product content in an enhanced e-book is primarily of the text type and has been enhanced with video and interactive graphics. The information is repeatable.

ONIX Code list for Product Content: **https://ns.editeur.org/onix/en/81)**

Example ONIX-30 Short-Format	Example ONIX-30 Reference-Format
<x416>10</x416> Eye-readable text	<PrimaryContentType>10</PrimaryContentType>

<b385>06</b385> Enhanced with video <b385>24</b385> Animated / interactive illustrations	<ProductContentType>06</ProductContentType> <ProductContentType>24</ProductContentType>

Table 4. Product Content

Extent

Extent covers product extents, in terms of pages, running times, file sizes, etc., as may be appropriate to each media type. The Extent type identifies the type of extent carried in the composite, e.g., running time for an audio or file size for an eBook product. The Extent unit indicating the unit used for the <ExtentValue> and the format in which the value is presented.

ONIX Code list for ExtentType: **https://ns.editeur.org/onix/en/23**

ONIX Code list for ExtentUnit: **https://ns.editeur.org/onix/en/24**

Example ONIX-30 Short-Format	Example ONIX-30 Reference-Format
<extent> <b218>22</b218> Filesize <b219>5000</b219> <b220>18</b220> Kbytes </extent>	<Extent> <ExtentType>22</ExtentType> <ExtentValue>5000</ExtentValue> <ExtentUnit>18</ExtentUnit> </Extent>

Table 5. Extent

E-publication technical protection (Digital Rights Management)

The information on Digital Rights Management (DRM) enables customers to inform whether or with which DRM protection a digital product is provided.

ONIX Code list for DRM: **https://ns.editeur.org/onix/en/144**

Example ONIX-30 Short-Format	Example ONIX-30 Reference-Format
<x317>02</x317> Digital watermarking	<EpubTechnicalProtection>02</EpubTechnicalProtection>

Table 6. Technical protection

Usage constraints

An optional group of data elements which together describe a usage constraint on a digital product (or the absence of such a constraint), whether enforced by DRM technical protection, inherent in the platform used, or specified by license agreement. Repeatable in order to describe multiple constraints on usage.

ONIX Code lists for usage constraints:

Usage type: **https://ns.editeur.org/onix/en/145**

Usage status: **https://ns.editeur.org/onix/en/146**

Usage unit: **https://ns.editeur.org/onix/en/147**

Example ONIX-30 Short-Format	Example ONIX-30 Reference-Format
<EpubUsageConstraint> <x318>05</x318> Text to speech <x319>01</x319> Is unrestricted </EpubUsageConstraint>	<EpubUsageConstraint> <EpubUsageType>05</EpubUsageType> <EpubUsageStatus>01</EpubUsageStatus> </EpubUsageConstraint>
<epubusageconstraint> <x318>03</x318> Copy/paste <x319>02</x319> Is limited <epubusagelimit> <x320>10</x320> Ten <x321>05</x321> Percent </epubusagelimit> </epubusageconstraint> <epubusageconstraint> <x318>06</x318> Lending <x319>02</x319> Is limited <epubusagelimit> <x320>1</x320> Only one <x321>10</x321> Occasion	<EpubUsageConstraint> <EpubUsageType>03</EpubUsageType> <EpubUsageStatus>02</EpubUsageStatus> <EpubUsageLimit> <Quantity>10</Quantity> <EpubUsageUnit>05</EpubUsageUnit> </EpubUsageLimit> </EpubUsageConstraint> <EpubUsageConstraint> <EpubUsageType>06</EpubUsageType> <EpubUsageStatus>02</EpubUsageStatus> <EpubUsageLimit> <Quantity>1</Quantity> <EpubUsageUnit>10</EpubUsageUnit> </EpubUsageLimit>

</epubusagelimit> <epubusagelimit> <x320>14</x320> For fourteen <x321>09</x321> Days </epubusagelimit> </epubusageconstraint>	<EpubUsageLimit> <Quantity>14</Quantity> <EpubUsageUnit>09</EpubUsageUnit> </EpubUsageLimit> </EpubUsageConstraint>

Table 7. Usage constrains

Product Form Feature and E-publication accessibility features for print-impaired readers

ONIX provides an opportunity to describe the accessibility features within e-books. That information can be transmitted in the Composite for Product form feature.

ONIX Code list for Product form feature type: **https://ns.editeur.org/onix/en/79**

ONIX Code list for E-publication Accessibility Details: **https://ns.editeur.org/onix/en/196**

Example ONIX-30 Short-Format	Example ONIX-30 Reference-Format
<productformfeature> <b334>03</b334> Text font <b336>18pt Tiresias LP</b336> </productformfeature>	<ProductFormFeature> <ProductFormFeatureType>03</ProductFormFeatureType> <ProductFormFeatureDescription>18pt Tiresias LP</ProductFormFeatureDescription> </ProductFormFeature>
<productformfeature> <b334>09</b334>E-publication accessibility detail <b335>21</b335> Text-to-speech hinting provided </productformfeature>	<ProductFormFeature> <ProductFormFeatureType>09</ProductFormFeatureType> <ProductFormFeatureValue>21</ProductFormFeatureValue> </ProductFormFeature>

Table 8. Product form feature and accessibility

2.7 Book crowdfunding and engagement with readers and writers

In the cultural-artistic world, we have seen a drastic change in the way the public consumes the products (let's just think about phenomena like Netflix, Spotify, Mubi, etc.). The editorial world has, on the contrary, kept a certain rigidity in regards to reader's role, which is still passive. This dynamic can be overturned by adapting the crowdfunding principles to the publishing industry, which give the reader an active, central role from the beginning.

Writers (first-timers, but also experienced) can encounter various difficulties when it comes to supporting oneself during the first steps of the launch process of their books, especially when they opt for self-publishing. It might sound like a better option, due to the apparent low-cost nature of self-promotion, but in reality, having a reliable publisher, with a clear market and strategy vision, makes all the difference when it comes to launching a first book or a new project. At the same time, incurring in the expensive (some might say, overpriced) fees of private consultants might not get the authors the results they work for.

Thus, the decision to implement crowdfunding campaigns – crowd-publishing, opens a way to improve writers' chances of getting published, without burdening them with financial obligations: an efficient crowdfunding campaign can cover up to a more than decent percentage of the costs (the remaining percentage is an investment made by the publisher), while providing the authors a reliable source of support, both material (through in-advance book reservations made by the readers), and immaterial (I.e., future readers, bookstores and online stores distribution, marketing campaign, etc.).

By pre-ordering copies for each project, the readers can actively choose what they actually want to read: a crowdfunding campaign allows the public to receive a preview of the chosen book right after the payment, and the final copy when the goal of the campaign is met.

Opposed to limiting oneself to exploring libraries, both on and offline, in a passive search of the next book to fall in love with, the participants in our crowdfunding campaigns actively select their readings.

The success of a crowdfunding campaign is largely based on WOMM (word-of-mouth marketing), as is the material success of books: by creating a 'Crowd', as the kind of environment that can motivate and create a strong, meaningful connection among readers and writers.

Future readers are part of the creative process: they put their trust in the writers by pre-ordering their works, they read and comment on the draft and can directly voice their opinions with the authors. As a result, new bonds are formed and new roles are created in our industry: every part is an active one, with decisional powers and influence.

The first, enthusiastic readers feel eager to share within their own environment/social bubble (both online and offline) their reading experience, working as an excellent amplifier. In this way, they gain ownership of the project, which is coming to life thanks to the support they are actually giving to the campaign. It is in their interest to help the book reach its goals, so a community is formed around the idea that the future book is the result of the shared effort of readers, authors and editors: the final customers aren't just consumers, but become a crucial asset in the creative process, to the point one might say that they are 'moral editors' of the book, and they know that.

At the same time, the authors create their own community, which does not limit its function to a single book, but becomes a solid base for the next projects too and, moreover, is an active type of community: the one that can give accurate feedback and help the author to grow. Of course, the first community the writers must refer to is the one formed by family and close friends, which at first might seem like a small one to rely on, but in reality, is the most supportive one, and it can kickstart the campaign with the right amount of enthusiasm.

Thanks to crowdfunding, the publisher has a lot to gain too: first of all, it is an excellent market test which guides the publishing choices based on the real preferences of the general public. This kind of insight makes it possible to work in a sustainable way, as the printed copies are exactly the ones requested, cutting down fixed costs and waste of paper and ink. Concurrently, WOM is indeed efficient for the publisher too: the publishing company improves its reputation amongst the communities and gets a strong and authentic bottom-up marketing support. Crowdfunding is a more inclusive, less biased and easier way to give aspiring authors an opportunity to express themselves, whilst the publishers take care of the quality of the submitted text, assuring that the published books are up to standards.

3 Möbius Envisioned Technology Outcomes

According to the DoA and to what emerged as relevant contributions to the sector from the users' perspective, Möbius will create different types of software applications:

- The Prosumer Intelligence Toolkit, composed of digital methods from computational social science in order to extract knowledge about collective dynamics of online fan communities that drive the production of value; the toolkit will include a dashboard for data exploration.
- The **Möbius Creators Toolkit** for helping professional users in the publishing industry to create and manage content in the new proposed format. The Creators' Toolkit is composed of three different modules to handle the different tasks when creating a Möbius book experience:
    - Data Dashboards, which provide a web-based tool for publishers to visualise the usage data gathered from the Möbius Player in a way that is meaningful and interactive.
    - Media Asset Manager, which will handle the different assets that make up a Möbius book (e.g., audio files, text, images) and facilitate the workflow necessary for creating the Möbius book (e.g., text and sound synchronisation, etc).
    - Spatial Audio Composer, a tool to position the narrator in 3D auditory scenes from existing monophonic content
- The Möbius Player, a dedicated application capable of displaying the Möbius book content to the end-users, allowing them to control the different interactive aspects

In the following subsections we present these envisioned outcomes in more detail.

3.1 Prosumer Intelligence Toolkit

The Prosumer Intelligence Toolkit (PIT) is a new prosumer intelligence framework and technology-enabled methods and tools based on data analytics that will provide the foundations for establishing effective cooperation with both publisher-managed and open communities of prosumers at all stages of the value chain. It solves the need of unprecedented scaling up of consumer intelligence, both in reach and scope for informing editorial decisions. Thus, it could be used for integrating user-driven approaches in workflows related to book publishing decisions, business models, and development of new products and services.

In practice the toolkit will consist of digital methods from computational social science in order to extract knowledge about collective dynamics of online fan communities that drive the production of value. The prosumer intelligence toolkit will leverage cutting-edge research on community dynamics and content co-creation in projects from the Wikimedia Foundation, e.g., Wikipedia, Wikidata.

Analysis techniques will include: social network analysis to identify user roles and influence, and to characterize community dynamics; text mining and temporal analysis to detect emerging trends; statistical analysis to compare different groups of users, works and topics based on different activity indicators; language and sentiment analysis to detect emotional patterns in content, discussions and reviews.

The target users of the PIT are publishers, and companies specialized in content creation; in particular, those who monitor markets and users to make editorial and/or strategic business decisions. For these target users the PIT will be an Interactive dashboard allowing for data exploration.

3.2 Möbius book

The Möbius book experience is a new concept enabling cross-media, interactive and immersive book experiences in the format for the digital publications, the EPUB standard. It will leverage interactive and immersive cross-media technologies to support social interactions, both physical and virtual.

Based on Möbius partner’s technologies, it is possible to develop a new concept enabling cross-media immersive and interactive book experiences. Starting from the print book, the Möbius enriches the book experience with cross media productions: (i) the digital version of the book, which include social interaction capacities (ii) the 3D audiobook for increased sense of immersiveness and enriched narrative by including three separate audio layers for narration, soundscapes and effects, and soundtracks, and (iii) audio-visual content based on the book arts, ad hoc artistic or marketing creations, or contributions from prosumers integrated into the narrative stream. Altogether, it will allow users to enjoy a cross-media reading experience or to cruise from one experience to another (e.g., print, eBook, audiobook) depending on their circumstances or wishes. By using 3D audio technologies, the Möbius book would also guarantee immersive experience for individual users and social experiences.

The Möbius book experience breakthrough is adding the sense of immersiveness by incorporating 3D audio and binaural reproduction to the reading experience. The minimum gear that an individual user needs to enjoy an immersive book experience is a regular mobile device (e.g., smartphones, tablets) or a desktop computer, as long as they have audio output. To do so, Möbius will include Ambisonics audio codification.

The figure below gives an overview of the building blocks of the Möbius book.

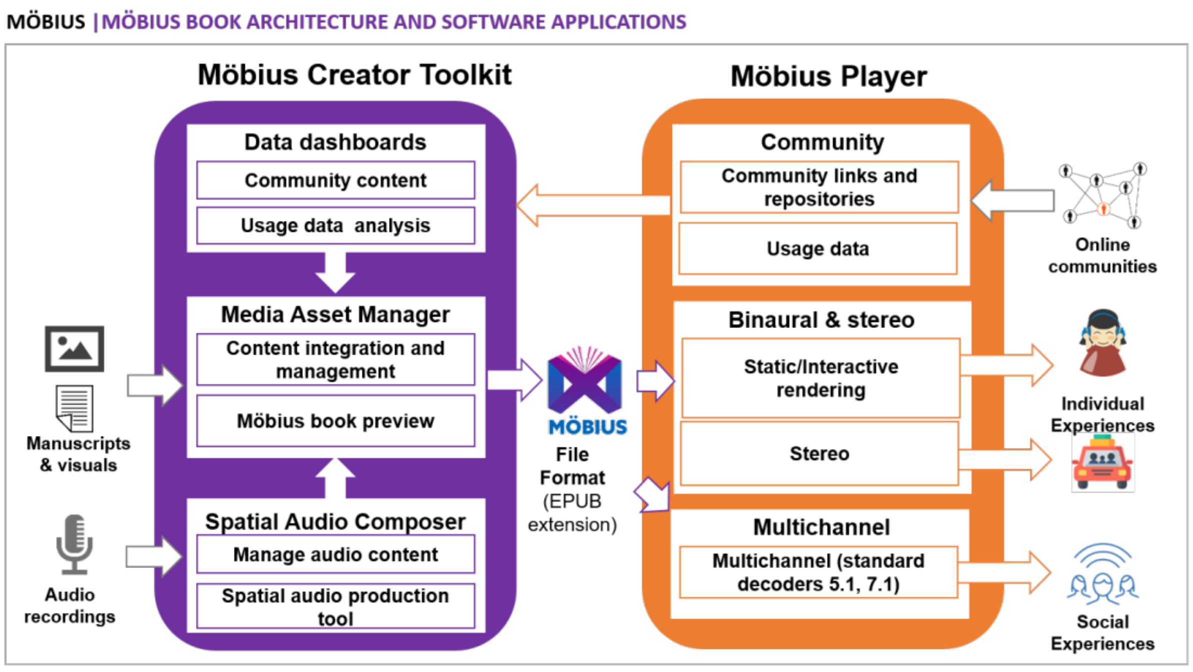


Figure 1.Möbius Book and software applications

3.2.1 Möbius Creator's Toolkit

The target users of the Creator's Toolkit are creative professionals, artists, content producers; in particular, those who create contents of their own, and/or following other's scripts.

The Möbius Creators Toolkit is composed of three different modules to handle the different tasks when creating a Möbius book experience:

- **Module 1. Data Dashboards**. This module provides a web-based tool for publishers to visualise the usage data gathered from the Möbius Player in a way that is meaningful and interactive (i.e. the charts displayed can be manipulated via user interactions). Thus, the user will be able to explore in detail the historical usage data related to each publication (e.g. average reading progress of users), conduct a comparative analysis between the data related to several publications, or have aggregate information related to all of their publications.

- **Module 2. Media Asset Manager.** Intended for the final publisher or editor, this module will handle the different components that will be in the final Möbius Book Format to be consumed by the Player. It allows non-technical users to add the relevant audio files, visualize and preview the contents of their book, and manage the metadata that will be integrated into the final production. It will also perform the process of forced alignment between the audio content and the transcript of the book, an automatic process in which time intervals for each word of the text is extracted from the audio signal. This forced alignment permits highlighting the words that are being played in the audio content and therefore enables the "Follow text" feature in the Player.
- **Module 3: Spatial Audio Composer.** This module is intended for the audio producers as a tool to position the narrator in 3D auditory scenes from existing monophonic content. The spatial audio composer will encode the audio in Ambisonics format allowing for subsequent audio rendering in static binaural (regular headphones), interactive binaural (head-tracking devices), stereo and multichannel systems. With the Spatial Audio Composer, the spatial soundtrack can be exported as a set of audio files in .wav format that are ingested by the Media Asset Manager or exported into other formats for its use in other types of productions (e.g., an immersive art installation as described in section)

3.2.2 Möbius Player

The Möbius Player is a desktop and mobile application responsible for delivering the content of a Möbius Book to the final user.

The target users of the Möbius Player are readers in general; in particular, those who enjoy audiobooks, and/or are open to experiencing books in unconventional ways.

The Player will include several modules:

- **Module 1. Community links and repository.** The Möbius Book will establish links with other readers and relevant online communities discussing the book or similar titles to allow interaction among them. The module will allow content creators to receive from the readers' original contents that can be used to enhance the Möbius Book experience. A media management backend will handle the uploaded content, transcoding, processing and indexing the received content so that it can be easily organised by the creators and users. A simple interface to contribute will enable this. Usage data about the way content is consumed (e.g., when, how much, on which modality, bookmarks, etc.) and the interactions will be collected and shared with the Möbius Creators Toolkit in order to provide key reader insights. Privacy-by-design methodology will be used such that this process is aligned with the GDPR. Data collected will be (pseudo)anonymised and the user will be able at any point to turn the data collection process on or off.
- **Module 2. Binaural & Stereo rendering.** This module will provide the rendering capabilities for the individual consumer. The Möbius Player will be aware of the playing conditions and automatically deliver the appropriate downmix from the Möbius Book Format. In mobile devices, or wherever the immersive audio is intended to be listened

through headphones, the process of converting the Ambisonic stream to binaural audio involves loading a SOFA compatible file that contains all necessary information for the two-channel binaural downmix. Both static and interactive binaural rendering will be supported depending on the head-tracking capabilities of the devices. This module will provide downmix to stereo for conventional stereo setups. The user will have the option to decide between three different modes: Reading only, Audio Only or Immersive.

- **Module 3. Multi-channel rendering.** This module is intended for the collective immersive experiences. In the case where a standard speaker layout is present, such as in common surround sound systems, the application will provide predefined decoders to downmix the ambisonic stream to the target systems (such as 5.1 or 7.1 layouts). For installations that target multiple listeners using non-standard loudspeaker configurations, a custom decoder will be generated by third party public tools and loaded to convert the ambisonic stream of the Möbius Book Format.

4 User Requirements

In this section we summarise the requirements as presented in Deliverable D2.1 with a view of further processing them in the light of the elaboration that is going to follow (RC: ranked by the consortium partners, RE-U: ranked by end-users, where a lower value means a higher rank).

Category	Description	RC	RE-U
Improve publishers' insights in reading and writing habits	**Möbius should bring publishers and readers closer together by helping publishers understand readers through data**	1	4
	Möbius should help publishers become more responsive to demands of users	2	3
	Möbius should provide publishers with behavioural data on how users engage with content	3	1
	Möbius should provide publishers with behavioural data on what is read	3	
	Möbius should provide publishers with behavioural data on how users prefer to read (device)	4	
	Möbius should provide publishers with insights on self-publishing based on user data (e.g., trends, liked content, used devices, time spent, etc.)	5	2

Discover new content	**Möbius should have a feature that recommends content to readers, based on their content**	6	**5**
	Möbius should have a feature that recommends content to readers, based on their community activity	7	7
	Möbius should have a feature that recommends content to readers, based on their interactions	8	6

Table 9. Relevant PIT user requirements

Category	**Description**	**RC**	**RE-U**
Improve publishers' insights in reading and writing habits	**Möbius should provide publishers with behavioural data on what is read**	**2**	**2**
	Möbius should provide publishers with behavioural data on how users prefer to read (device)	**3**	
Improve the publishing process as a whole	Möbius should help publishers reduce costs by automating conversion to audio	14	11
	Möbius should help publishers go green by improving format and distribution	6	**4**
Engage users in the publishing process	**Möbius should allow publishers to engage prosumers in the publishing process**	7	**1**
	Möbius should help publishers to improve the publishing process by establishing a direct relationship with readers	8	**3**
Finding the (right) audience & interaction	Möbius should allow users to add tags to content (e.g., 'young adults') so other readers can select stories easily	10	10
	Möbius should allow authors to select which categories of user feedback to display	15	15
Self-promotion	**Möbius should support writers to promote their work, by helping them to define useful keywords (SEO, SEA)**	**5**	12

	Möbius should support users willing to use promotional tools (existing tools–or Möbius could create promotional tools)	12	14
	Möbius should allow users to easily share their content (as a package: text, video, audio, etc.)	**1**	6
Writing skills	Möbius should help users to find existing tools to improve their writing skills	14	13
	Möbius could suggest word and sentence improvements, based on existing popular fiction	16	9
	Möbius should offer a toolkit that improves users' creative writing (based on both fiction genres and audience)	13	**5**
	Möbius should offer an AI tool that scans fanfic to not interfere with existing IP	9	10
Multimedia	**Möbius should allow users to create an immersive multimedia experience, by adding audio-visual content to their text**[102]	**4**	8
	Möbius should support combined reading and listening consumption (like Amazon), the audio could add extra characteristics to the text	**2**	7

Table 10. Relevant Möbius Book user requirements

Category	Description	RC	RE-U
Finding the (right) audience & interaction	**Möbius should have a network feature that connects writers and readers with similar interests**	**5**	**2**
	Möbius should allow users to leave feedback just for the author via likes in different categories (such as true to character, plot, style, grammar)	7	**3**
	Möbius should provide a place where literature (fanfic, prosumer, self-published, etc.) can be discussed in a slow pace (e.g., via traditional forum)	11	/

Writing skills	Möbius should allow users to like (or review) other users' stories and writing skills separately	8	7
	Möbius should allow users to contribute to grammatical corrections as a community	11	8
	Möbius should allow users to disclose their texts for modification by others, to moderate and approve other users make to their texts changes	8	/
Multimedia	**Möbius should provide a platform where users can engage with new types of content (not just eBook or audiobook)**	**4**	**5**
	Möbius should provide an immersive or enhanced experience that is accessible to a large audience (beyond tablet or eBook)	6	**4**
	Möbius should provide additional content that helps the reader understand or live the story	**3**	/
	Möbius should allow audio to play while reading a section or chapter	6	/
	Möbius should allow users to link to websites related to fan fiction stories (e.g., clothing, interior, travel...)	9	9
	Möbius should allow users to explore content like videogames (e.g., visit different parts of a room, without following a fixed story line)	10	10
	Möbius should allow users to continue 'reading' a story by switching from reading mode to audiobook	**2**	11
Discover new content	**Möbius' users (both readers and writers) should be able to indicate the quality of content (e.g., likes, comments, shares, etc.)**	**1**	**1**
	Möbius should allow authors/writers and users to indicate how close the fan fiction is to the original character(s)	12	6

Table 11. Relevant Möbius Book Player user requirements

5 Functional and system requirements

In this section we synthesize the requirements above with a view to group together related functionality, discuss their implications and feasibility regarding their technical implementation and turn them in a list of system requirements for each of the Möbius technological outcomes. In the assessment of the requirements, we will break down their effect on backend, frontend and storage, a common way to technically separate business logic interfaces and data models. Furthermore, in this analysis we will assess the requirements using the already identified broader categories of them. For each of these categories we will identify components assessing backend, frontend and storage requirements. This is a simplified way to define components using the Model-View-Controller[37] (MVC) design pattern. This design pattern allows to separate the program logic in three elements:

- **Models**: The dynamic structures that hold information about data (these structured can be derived when discussing *Storage* below)
- **Views**: The different representations of the models as part of a user interface (these representations can be derived when discussing *Frontend* below)
- **Controllers**: The elements that accept user input and transform user actions into models and views (these actions can be derived when discussing *Backend* below).

Under these considerations, the next sections analyse each requirement category in detail.

5.1 Prosumer Intelligence Toolkit

5.1.1 Improve publishers' insights in reading and writing habits

The table below presents the user requirements for the Prosumer Intelligence Toolkit in the category of "Improve publishers' insights in reading and writing habits" as they have been elaborated in D2.1 and servers as a basis for our considerations on the related functional requirements.

#	Requirement	Rank by Consortium	Rank by End-Users
R1	Möbius should bring publishers and readers closer together by helping publishers understand readers through data	1	4
R2	Möbius should help publishers become more responsive to demands of users	2	3
R3	Möbius should provide publishers with behavioural data on how users engage with content	3	1

[37]Model-View-Controller, https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

R4	Möbius should provide publishers with behavioural data on what is read	3	
R5	Möbius should provide publishers with behavioural data on how users prefer to read (device)	4	
R6	Möbius should provide publishers with insights on self-publishing based on user data (e.g., trends, liked content, used devices, time spent, etc.)	5	2

Table 12. Prosumer Intelligence Toolkit (Publishers' insights in reading and writing)

These requirements refer to the Prosumer Intelligence Toolkit where the main data sources are prosumer communities. From a functional perspective, R1 is a more general requirement that represents the vision of this component. The other set of requirements (R2, R3, R4, R5 and R6) presumes a way to capture actions in the communities of focus. Capturing these properties though depends on a) their availability in the respective community and b) on their programmatic or otherwise possible access to them.

The corresponding part of the Möbius system functional module can therefore be specified as having the following components:

- Backend
  - Capture, extract and pre-process properties from prosumer communities
  - Integrate them as query parameters as part of the knowledge extraction models and main developments in T3.1
- Frontend
  - Construct a query so that users can request specific properties along defined time dimensions. The query interface should provide a list of predefined properties (from T3.1) also a list of predefined time intervals (e.g., last 24 hours, last week, last month)
  - Visualise time-based data as charts. Possible chart types are bar, pie and line charts, or gauge-type visualisations
- Storage
  - Store queries and query results as part of the user profile in a database table

5.1.2 Discover new content

The table below presents the user requirements for the Prosumer Intelligence Toolkit in the category of “Discover new content” as they have been elaborated in D2.1 and servers as a basis for our considerations on the related functional requirements.

#	Requirement	Rank by Consortium	Rank by End-Users

R7	Möbius should have a feature that recommends content to readers, based on their content	6	5
R8	Möbius should have a feature that recommends content to readers, based on their community activity	7	7
R9	Möbius should have a feature that recommends content to readers, based on their interactions	8	6

Table 13. Prosumer Intelligence Toolkit (Discover new content)

This functionality builds up on the module that stores actions of users and extends the Möbius system with tools to provide recommendations to users. First it is important to note that the type of recommendations could also work for a limited number of Möbius books in the system but require a large amount of user activity and interactions so that they are more relevant to the end users. Secondly this requirement refers to the content processed from the fanfiction communities through the Knowledge Extraction Models part of the Prosumer Intelligence Toolkit. It is not clear to which extent there will be continuous access to this community content and how often this content could be updated. The recommendations however can be provided independently of this, even based on content that is not as actual.

From a first assessment of the requirements, we see that R7 and R8 refer to similar functionality namely the one related to user interactions. Community activity in this sense is an aggregated view of the user interactions within a specific context.

The type of recommendations mentioned in R7, R8/R9 can be seen to be generally as a type of collaborative filtering.[38] Thus, the recommendation problems can be rephrased as:

- R7: “Given what other readers have read, show me the most related book titles that partially match also the book titles I have read”.
- R8: “Given the interactions of other readers in my community, show me the most related book titles that partially match my interactions in this community”
- R9: “Given the interactions of other readers, show me the most related book titles that partially match my interactions.

The main components of the Möbius system module dealing with this functionality are then:

- Backend
  - A service running the recommendation algorithm, taking as input the relevant entities of above (I.e., book titles read, user interactions within a community, user interactions in general). Depending on the number of users and interactions this service could be time consuming and should therefore run as a cron job (i.e., in specified intervals) for example once every 24 hours.

[38]Collaborative filtering is a technique used by recommender systems, https://en.wikipedia.org/wiki/Collaborative_filtering

- Frontend
    - The frontend could be an entry of the last recommendation in the user profile or in the user home page. In this case we plan to show the newest item in the recommendation result list for the active user.
    - We could also think of presenting recommendations as notifications. In this case we could show only the newest item from the recommendation result list that has not been shown yet to the active user.
    - Additionally, recommendations could also be available under a specific menu action. When available in this way we could show up to 3 items for each recommendation category.
- Storage
    - Store separately the input to the recommendation service (3 stores, one for each of the categories above) and the output of the recommendation service for each user in a database table

5.2 Möbius Book Creator's Toolkit

5.2.1 Improve publishers' insights in reading and writing habits

The table below presents the user requirements for the Möbius Creators' Toolkit in the category of "Improve publishers' insights in reading and writing habits" as they have been elaborated in D2.1 and servers as a basis for our considerations on the related functional requirements.

#	Requirement	Rank by Consortium	Rank by End-Users
R10	Möbius should provide publishers with behavioural data on what is read	2	2
R11	Möbius should provide publishers with behavioural data on how users prefer to read (device)	3	

Table 14. Möbius Book Creator's Toolkit (Publishers' insights in reading and writing)

Requirements R10 and R11 are similar in essence to R4 and R5 respectively but they refer to a different source, namely the Möbius book. These requirements also presume that there is a way to capture actions by users, aggregate them into a data space and provide useful visualisations to present historical charts and infer trends. The main properties that could be captured are:

- time a user is reading,
- clicks of users, e.g., likes
- devices of users

It is worth to be noted that these requirements require users to opt-in and agree that the Möbius system captures these properties.

The corresponding part of the Möbius system functional module can therefore be specified as having the following components:

- Backend
    - Methods to manage opt-in preferences (e.g., allow capture for opted-in users, delete captured properties for opted-out users)
    - Capture, extract and pre-process the requested properties in real-time as they happen for opted-in users
- Frontend
    - Opt-in form per user, as part of the user registration, change of opt-in as part of a user preference page
    - Construct a query so that users can request specific properties along defined time dimensions. The query interface should provide a list of predefined properties (e.g., time read, clicks. devices) also a list of predefined time intervals (e.g., last 24 hours, last week, last month)
    - Visualise time-based data as charts. Possible chart types are bar, pie and line charts, or gauge-type visualisations
- Storage
    - Store the requested properties as part of the user profile and additionally in an anonymised aggregated form as time-series data in a database table
    - Depending on the number of concurrent users and interactions, a heavy load can be placed on the database. In order to mitigate this risk, we could use an in-memory key store (e.g., memcached[39] or redis[40]) as a temporary cache.

5.2.2 Improve the publishing process as a whole

The table below presents the user requirements for the Möbius Creators' Toolkit in the category of "Improve the publishing process as a whole" as they have been elaborated in D2.1 and servers as a basis for our considerations on the related functional requirements.

#	Requirement	Rank by Consortium	Rank by End-Users
R12	Möbius should help publishers reduce costs by automating conversion to audio	14	11
R13	Möbius should help publishers go green by improving format and distribution	6	4

Table 15. Möbius Book Creator's Toolkit (Publishing process)

The realisation of R12 requires a component to convert text to speech (TTS, Text-to-Speech, also known as Speech synthesis). Speech synthesis is defined as the artificial production of

[39]Memcached, high-performance, distributed memory object caching system, https://www.memcached.org
[40]Redis, an in-memory data structure store, https://redis.io/

human voices. TTS has come a long way in the last years and improved a lot to text synthesis has come a long way and there are many use cases, especially in the area of accessible content processing (e.g., in screen readers for reading aloud text in a website for visually impaired persons). The quality of the synthesis depends on the availability of synthetic voices and how well these are synthesised in natural sounding phrases that take into account punctuation, emphasis and can convey other emotions. The current state of the art does not allow for a replacement of a human narration though. In Möbius we could connect to external APIs e.g., Google Text-to-Speech[41] and provide methods to convert text into audio.

The requirement R13 can be satisfied by primary producing digital formats and distributing them in a digital way along with improvements in efficiency of production. Since Möbius books are inherently digital, their production (i.e., the Möbius Creator's Toolkit) and their distribution (i.e., the Möbius Player) are digital too, in order to cover this requirement, improvements in production efficiency would require a baseline to compare the ways content is produced without and with the Möbius tools. Möbius could provide additional statistics on the number of books produced, maybe also including the total number of pages in these books and other relevant data like production efficiency for publishers to calculate their carbon footprint, respectively savings due to Möbius.

From a functional perspective R12 and R13 differ so we will examine them separately. With regards to R12 a Möbius component could have

- Backend
  - Connection to third party APIs for Text-to-Speech synthesis that includes sending a specified text excerpt and receiving the corresponding audio
  - A method to match text excerpt and audio, aligning text phrase and time-code
- Frontend
  - An extended audio player able to play the audio time-codes and highlight the corresponding text in a synchronised manner
  - An audio player able to play the audio-time codes for use-cases related to audio-books only
- Storage
  - A fast read/write file storage for the corresponding audio files that allows for adequate provision of bandwidth (e.g., to cater for concurrent users requesting audio files)
  - Store identifiers to text synthesized and corresponding identifiers to audio files for each book title

R13 could be part of a dashboard/analytics component

- Backend
  - A method to calculate number of books per publisher, aggregate number of pages and other relevant metrics
- Frontend
  - An entry in a dashboard showing the corresponding metrics

[41]Google Text-to-Speech, https://cloud.google.com/text-to-speech/

- Storage
    - Store for each publisher the identifiers of each book and its number of pages in a database table

5.2.3 Engage users in the publishing process

The table below presents the user requirements for the Möbius Creators' Toolkit in the category of "Engage users in the publishing process" as they have been elaborated in D2.1 and servers as a basis for our considerations on the related functional requirements.

#	Requirement	Rank by Consortium	Rank by End-Users
R14	Möbius should allow publishers to engage prosumers in the publishing process	7	1
R15	Möbius should help publishers to improve the publishing process by establishing a direct relationship with readers	8	3

Table 16. Möbius Book Creator's Toolkit (Engage users in the publishing process)

Engagement in R14 can be seen as messaging or getting in contact with prosumers (e.g., through a provided chat, using a contact facility, or using a direct email) during the different stages of the publishing process. In addition, also a way to request feedback, possibly using a third-party form creation service[42] could be considered. R15 could be fulfilled by the same technical means as in R14. It should be noted that both prosumers and readers need to opt-in to this specific functionality.

From a functional perspective these requirements introduce the following components:

- Backend
    - Methods to process contact forms, notify respective actors and store results per user
    - Methods to send chat-like messages and notifications
    - Methods to capture interactions between prosumers/users and publishers
    - Community building methods like connect to a publisher, connect to a user
    - Methods to list established connections
- Frontend
    - Opt-in interface as part of the registration process and as an additional setting in the user preferences
    - Fill contact form, validate form fields
    - Interface to send and receive chat-like messages together with missing notifications
- Storage

[42]For example, Tally https://tally.so/, SurveyMonkey https://www.surveymonkey.com/

- Store messages as part of the user interactions in a database table together with their delivery status (e.g., sent, pending) and message modality (e.g., contact form, chat)
- Store contact details and opt-in preferences of prosumers, users and publishers in a database table

5.2.4 Finding the (right) audience & interaction

The table below presents the user requirements for the Möbius Creators' Toolkit in the category of "Finding the (right) audience & interaction" as they have been elaborated in D2.1 and servers as a basis for our considerations on the related functional requirements.

#	Requirement	Rank by Consortium	Rank by End-Users
R16	Möbius should allow users to add tags to content (e.g., 'young adults') so other readers can select stories easily	10	10
R17	Möbius should allow authors to select which categories of user feedback to display	15	15

Table 17. Möbius Book Creator's Toolkit (Audience and Interaction)

R16 extends the Möbius data model about books with a property called "tags". These tags should be user-facing and could be used as a search filter. Tags should be added only by authorised users, namely the owners of the respective book. In order to cater for many tags R16 also introduces a backend component for full-text search indexing and filter-based searches (e.g., Apache Solr[43]) to allow users to efficiently query based on tags. Tags could be provided as an autocomplete suggestion when a user started entering the first three characters and also as a part of a select dropdown input field

R17 introduces an approval process for categories of feedback received. This also assumes that feedback is organised in categories as well that there is a way to limit feedback based on the selected organised categories like character, plot, style and grammar. Feedback can have the form of a reaction (e.g., a like in an emoji style "thumbs up") and optionally include a text field.

From a functional perspective R16 and R17 are different, so we will elaborate on their implications individually below.

For R16 the effect on the functional requirements consists of:

- Backend
  - Methods to provide autocomplete suggestions for example after three characters are typed in the tags input field

[43]Apache Solr, https://solr.apache.org/

- Methods to provide the complete list of tags
    - Methods to index tags as part of a full-text and filter-based search index
    - Methods to search and filter based on one or more tags
- Frontend
    - Input form to search for books which hold one, or more tags. The input form could provide autocomplete suggestions.
    - Input form to add one, or more tags to a book. The input form could provide autocomplete suggestions.
- Storage
    - Extension of the book data model with an additional property "tags" property. Tags could be a list of terms that are shared among users

For R17 the following components should be added to the Möbius system

- Backend
    - Methods to select allowed categories from a predefined list of feedback categories
    - Methods to store feedback on the specific book item under the respective category
    - Methods to retrieve feedback of a book from the allowed categories
- Frontend
    - A component to add feedback that includes a emoji-style reaction and an optional text field
    - A component to show add feedback and the category it belongs
- Storage
    - Store feedback properties (category reaction, text) referencing the book identifier and the user identifier in a database table
    - Store predefined categories of feedback in a database table
    - Store predefined reactions for feedback in a database table

5.2.5 Self-promotion

The table below presents the user requirements for the Möbius Creators' Toolkit in the category of "Self-promotion" as they have been elaborated in D2.1 and servers as a basis for our considerations on the related functional requirements.

#	Requirement	Rank by Consortium	Rank by End-Users
R18	Möbius should support writers to promote their work, by helping them to define useful keywords (SEO, SEA)	5	12

R19	Möbius should support users willing to use promotional tools (existing tools–or Möbius could create promotional tools)	12	14
R20	Möbius should allow users to easily share their content (as a package: text, video, audio, etc.)	1	6

Table 18. Möbius Book Creator's Toolkit (Self-promotion)

Search Engine Optimisation (SEO) is primarily aiming to improve the position of an entry in a search result. Since Google is currently the search engine with the largest reach SEO Is aiming to move results up the Google list. This entails a) making content easier to crawl (and by extension understand) by the GoogleBot[44] and b) the use of appropriate metadata. In the context of Möbius we could try to take advantage of structured data[45] (among them, the Book[46], and the Article[47] schema), however the use of this metadata is regulated by Google and access to them can be denied for any reasons. It should also be noted that these structured metadata are only available for web-based file formats (e.g., HTML) and not directly in EPUB.

Search Engine Advertising (SEA) involved creating paid ads on search engine. This is mainly an action outside of Möbius and is handled by the search engines themselves through dedicated applications[48]. There are APIs available for e.g., Google AdWords[49] but an integration of them into Möbius is out of scope.

R18 could be thus mainly addressed through the use of structured data.

With regards to R19, additional promotional tools could be the creation of mailing lists per book/author of interested readers, the sending of newsletters, promotional offers or other marketing communication. This requirement requires readers to opt-in in this marketing-type of communication. There are many existing tools[50] that can be used for this type of marketing type communication that already handle the creation of mailing list, segmentation of target audiences and sending of large e-mail campaigns. Möbius could support this process by allowing an export of audience data in a format that can be easily imported into these tools. From a first analysis it seems that a format using comma-separated values or otherwise known as CSV[51] is best suitable for this purpose.

[44]GoogleBot, https://developers.google.com/search/docs/advanced/crawling/googlebot
[45]Understand how structured data works, https://developers.google.com/search/docs/advanced/structured-data/intro-structured-data
[46]Book Schema, https://developers.google.com/search/docs/advanced/structured-data/book
[47]Article, https://developers.google.com/search/docs/advanced/structured-data/article
[48]Microsoft Advertising (formerly Bin Ads), https://about.ads.microsoft.com/ and Google AdWords https://ads.google.com/
[49]Google Ads API, https://developers.google.com/google-ads/api/docs/start
[50]For example, Mailchimp https://mailchimp.com/ or SendInBlue https://www.sendinblue.com/
[51]Comma-Seperated-Values, https://en.wikipedia.org/wiki/Comma-separated_values

R20 requires the provision of an asset management system that is capable to hold different types of content, access file storage and provide streaming and download capabilities.

From a functional perspective all three requirements are different and we will discuss their implications individually below.

For R18 Möbius would require the following components:

- Backend
  - Methods to create the appropriate structured metadata for each type of content
  - Methods to render structured metadata in HTML pages
- Frontend
  - Interface to add any structured metadata that are not part of the book/author data model
- Storage
  - Store additional structured metadata, that are not captured previously as part of the book/author data model

The main Möbius component for R19 is a way to opt-in and an export of audience data to a CSV format:

- Backend
  - Methods to handle and record opt-in of users
  - Methods to export user data per book and/or author in a CSV format
- Frontend
  - Interface to opt-in, probably as part of the registration process, together with an option to change this setting in a preference pane.
  - Interface to request the CSV export
- Storage
  - File storage of CSV exports with additional downloading facilities.
  - Store identifiers to exports and file locations in the database table

R20 requires Möbius to provide a complete asset management functionality consisting of:

- Backend
  - Methods to process, store, transcode and provision audio files in different resolutions
  - Methods to process, store, transcode and provision video files as a stream in different resolutions or as objects to be downloaded
  - Methods to process, store and provision images in different resolutions
  - Methods to process, store and add text or text-based documents
  - Methods to export book related files
- Frontend
  - Interface to upload files, add file descriptions and tags
  - Interface to manage (e.g., edit/delete) and to link files to specific segments of books

-
    - Interface to export all book related files in an EPUB container,[52] which essentially is a single unencrypted zipped archive containing all book resources
- Storage
    - A fast IO file storage, potentially under the same book folder
    - Store identifiers of files, their locations, other file metadata along with book identifiers in a database table.

5.2.6 Writing skills

The table below presents the user requirements for the Möbius Creators' Toolkit in the category of "Writing skills" as they have been elaborated in D2.1 and servers as a basis for our considerations on the related functional requirements.

#	Requirement	Rank by Consortium	Rank by End-Users
R21	Möbius should help users to find existing tools to improve their writing skills	14	13
R22	Möbius could suggest word and sentence improvements, based on existing popular fiction	16	9
R23	Möbius should offer a toolkit that improves users' creative writing (based on both fiction genres and audience)	13	5
R24	Möbius should offer an AI tool that scans fanfic to not interfere with existing IP	9	10

Table 19. Möbius Book Creator's Toolkit (Writing skills)

R21 is referring to existing tools that improve writing skills. However, as these tools are not yet defined it is impossible to assess and design the relevant Möbius functional components. For this reason, we consider this requirement as out-of-scope at this time and we will revisit it later when more information from the pilot phase is available.

R22 and R23 are related and can be seen primarily as a fuzzy search of content that is available in Möbius. Looking at it from this angle, the search query would consist of a phrase and the result would provide similar phrases used. Additionally, search results can be filtered based on genre and audience to provide a more focused outcome. The requirement requires the definition of a suitable similarity metrics. Given this similarity metrics, using Apache Solr (see R16) Möbius could provide a way to satisfy this requirement.

Since access to content from the fanfic communities is only provided by the Prosumer Intelligence Toolkit, this requirement relates more to this Möbius tool. It is however questionable at the moment to which extent there could be a quasi-live access to fanfic

[52]EPUB, https://en.wikipedia.org/wiki/EPUB

communities'' content or how often this content will be updated by the Prosumer Intelligence Toolkit. The requirement can maybe be met for a specific snapshot of fanfic community content but the value of any result is limited, thus we consider this out-of-scope for Möbius.

For R22 and R23 the possible functional components of Möbius are:

- Backend
  - Methods to search similar text content having as input a text phrase
- Frontend
  - Interface to highlight a text segment (possibly limited to a maximum number of characters) and initiate a similarity search
  - Interface to show the results of the similarity search
- Storage
  - Store additional search indexes in Apache Solr

R24 is referring to a copyright infringement checker. This implies the availability of a complete database of IP works which can be indexed. Such a tool is however outside the scope of the project.

5.2.7 Multimedia

The table below presents the user requirements for the Möbius Creators' Toolkit in the category of "Multimedia" as they have been elaborated in D2.1 and servers as a basis for our considerations on the related functional requirements.

#	Requirement	Rank by Consortium	Rank by End-Users
R25	Möbius should allow users to create an immersive multimedia experience, by adding audio-visual content to their text	4	8
R26	Möbius should support combined reading and listening consumption (like Amazon), the audio could add extra characteristics to the text	2	7

Table 20. Möbius Book Creator's Toolkit (Multimedia)

R25 and R26 refer functionally to similar components of the Möbius system that deal with adding 3D audio content together with additional multimedia files (photos, videos, documents) and enrich the available text. The 3D audio should be synchronised with the text and provide a combined reading and listening experience. Photos, videos and other files will be added as pointers or links to specific text segments.

The main components of the Möbius tools to satisfy these requirements are given below. We differentiate here between audio narration and 3D-audio content which potentially includes special audio effects.

- Backend
  - Tools to produce audio narration
  - Tools to produce 3D-audio content including audio effects
  - Methods to store audio narration, 3D-audio content, and other multimedia files
  - Methods to align audio narration and 3D audio content with text segments
- Frontend
  - Interface to produce audio narration (start/stop recording)
  - Interface to aid the production of 3D content, e.g., for setting the sound field and sound directions
  - Interface to upload audio narration, 3D-audio, and other multimedia files
  - Interface to align 3D-audio effects, photos, videos to specific text segments
  - Interface to show the existence of available additional information
- Storage
  - File storage for all assets (see also R20) as part of a book entity

  - Store asset identifiers and related book identifiers in a database table

5.3 Möbius Book Player

5.3.1 Finding the (right) audience & interaction

The table below presents the user requirements for the Möbius Book Player in the category of “Finding the (right) audience & interaction” as they have been elaborated in D2.1 and servers as a basis for our considerations on the related functional requirements.

#	Requirement	Rank by Consortium	Rank by End-Users
R27	Möbius should have a network feature that connects writers and readers with similar interests	5	2
R28	Möbius should allow users to leave feedback just for the author via likes in different categories (such as true to character, plot, style, grammar)	7	3
R29	Möbius should provide a place where literature (fanfic, prosumer, self-published, etc.) can be discussed in a slow pace (e.g., via traditional forum)	11	/

Table 21. Möbius Book Player (Audience and Interaction)

R27 refers to a feature recommending interested or related users to users, mainly writers and readers, but it could be extended to include writers to writers and readers to readers. This is essentially a recommendation problem based on user profile similarity. User profile similarity can be expressed taking account interests and interactions. R27 is an extension of R7, R8 and R9 already dealt above, adding an additional recommendation entity I.e., user profiles but does not essentially alter the Möbius functional components presented there.

R28 is related to R17 and does not introduce any special additional changes to Möbius components but only exemplifies possible feedback categories.

R29 introduces a forum or a message board as a means of discussion and interaction modality of the different actors involved. The Möbius functional requirements are then:

- Backend
  - Methods to add forum entries, possibly with an upvoting feature and insertion of tags and uploading of photos, and audio-visual files
  - Methods to index forum entries as part of the search index
  - Methods to search based on full-text, filter based on tags
  - Methods to mark discussions as active or stale based on recency and number of entries in a predefined time period.
  - Methods to aggregate tags and provide means to explore content, e.g., using a tag cloud visualisation
  - Methods to add replies and organise discussions in threads
- Frontend
  - Interface to add forum entries
  - Interface to show active discussions
  - Interface to search discussions based on tags and display results
  - Interface to show threaded discussions
- Storage
  - Store forum entry identifiers, text, tags, file identifiers, replies and thread identifiers in a database table
  - Store file identifiers, entry identifiers and file locations in a database table
  - File storage for related photos and audio-visual files

5.3.2 Writing skills

The table below presents the user requirements for the Möbius Book Player in the category of "Writing skills" as they have been elaborated in D2.1 and servers as a basis for our considerations on the related functional requirements.

#	Requirement	Rank by Consortium	Rank by End-Users
R30	Möbius should allow users to like (or review) other users' stories and writing skills separately	8	7
R31	Möbius should allow users to contribute to grammatical corrections as a community	11	8

R32	Möbius should allow users to disclose their texts for modification by others, to moderate and approve other users make to their texts changes	8	/

Table 22. Möbius Book Player (Writing skills)

R30 deals with giving feedback on certain feedback categories and is similar in this respect to R28 and R17 and is not introducing any changes in the already devised components.

R31 and R32 are related with regards to their functional implementation and introduce an editorial process in Möbius. This process can be seen as an active request for comments from authors. In this respect the functional implementation of these requirements in Möbius is:

- Backend
  - Methods to create a dedicated hashed link for a book. Users with the link can comment and review but not edit content
  - Methods to invite reviewers for providing comments and suggestions
  - Methods to attach comments to specific text segments
  - Methods to accept comments and suggestions
- Frontend
  - Interface for reviewers to select a specific text segment and add comment
  - Interface for authors to accept comments
- Storage
  - Store book identifier, user identifiers and their comments on specific text segments and the current status of the comment (e.g., accepted, rejected) in a database table

5.3.3 Multimedia

The table below presents the user requirements for the Möbius Book Player in the category of "Multimedia" as they have been elaborated in D2.1 and servers as a basis for our considerations on the related functional requirements.

#	Requirement	Rank by Consortium	Rank by End-Users
R33	Möbius should provide a platform where users can engage with new types of content (not just eBook or audiobook)	4	5
R34	Möbius should provide an immersive or enhanced experience that is accessible to a large audience (beyond tablet or eBook)	6	4

R35	Möbius should provide additional content that helps the reader understand or live the story	3	/
R36	Möbius should allow audio to play while reading a section or chapter	6	/
R37	Möbius should allow users to link to websites related to fan fiction stories (I.e., clothing, interior, travel)	9	9
R38	Möbius should allow users to explore content like videogames (e.g., visit different parts of a room, without following a fixed storyline)	10	10
R39	Möbius should allow users to continue 'reading' a story by switching from reading mode to audiobook	2	11

Table 23. Möbius Book Player (Multimedia)

R33, R34, R35 are similar in regard to their functional implementation with R25 and don't introduce any new Möbius components from those already presented.

R37 extends R25 with the option to add links to specific text segments of a book. To this effect the change in the Möbius components is:

- Backend
    - Methods to connect text segments of a book with external links
- Frontend
    - Interface to show the existence of these additional links
- Storage
    - Store link identifiers and link locations on text segments together their book identifiers in a database table

R36 and R39 are similar to R26 but introduce the capturing of the current reading state to be able to switch from one modality to the other. The resulting Möbius functional component is then:

- Backend
    - Methods to automatically capture reading state, possibly on a page level if only the text modality is selected, and to the segment level is a combined text/audio modality is selected
    - Methods to manually initiate the change of the reading state
- Frontend
    - Interface to manually capture the reading state e.g., "Continue on the audio mode"
- Storage

    - Store text segment, page of the current reading state for each user and book in a database table

R38 introduces a non-linear way of navigation in Möbius books. This can be accomplished to a large extent with the additional files and links that are attached to text segments. The navigation provided is mainly hierarchical (e.g., from a text segment on a book to an associated asset and back), Möbius could easily implement navigation across the available assets and add non-linear navigation components. The related functional components extend those provided R25 and R26 in the following way:

- Backend
    - Methods to list all assets of a book, potentially groups by type (e.g., photo, video)
- Frontend
    - Interface to initiate the display of all available assets when a user opens additional information associated to a text segment
    - Interface to display all available assets when a user requests it
- Storage
    - No specific new storage requirements

5.3.4 Discover new content

The table below presents the user requirements for the Möbius Book Player in the category of "Discover new content" as they have been elaborated in D2.1 and servers as a basis for our considerations on the related functional requirements.

#	Requirement	Rank by Consortium	Rank by End-Users
R40	Möbius' users (both readers and writers) should be able to indicate the quality of content (e.g., likes, comments, shares, etc.)	1	1
R41	Möbius should allow authors/writers and users to indicate how close the fan fiction is to the original character(s)	12	6

Table 24. Möbius Book Player (Discover new content)

R41 is adding an additional feedback category and is regarding its functional implementation similar to R17 not adding any new functional requirements.

R40 caters also for an additional feedback mechanism and a way to display a user-generated quality score. Potentially it also implies a way to sort content based on this quality score. It extends the functional requirements described in R17 in the following way:

- Backend

- Methods to calculate a content quality score based on user-ratings and feedback
- Methods to sort by quality score

- Frontend
  - Interface to show the sorted results
- Storage
  - Store quality score and its evolution across time for each book in a database table

5.4 Summary of system requirements

Below we summarise the system requirements that were described in the previous sections, grouping them together.

In the notation we use the following convention:

- ”B” = Backend,
- “F” = Frontend,
- “S” = Storage.

An overall priority of the requirements is given on a scale from 1 to 3 (1 being the highest) taking into account the underlying assigned “ranked by consortium” and “rank by user” as well as the role that the requirement plays for the overall system architecture.

5.4.1 System requirements of Prosumer Intelligence Toolkit

#	Requirement	Priority
B1	Capture, extract and pre-process properties from prosumer communities	1
B2	Integrate properties from prosumer communities as query parameters (as part of the knowledge extraction models and main developments in T3.1)	1
B3	A service running the recommendation algorithm, taking as input the relevant entities of above (I.e., book titles read, user interactions within a community, user interactions in general). Depending on the number of users and interactions this service could be time consuming and should therefore run as a cron job (i.e., in specified intervals) for example once every 24 hours.	2

Table 25. Backend requirements of Prosumer Intelligence Toolkit

#	Requirement	Priority
F1	Construct a query so that users can request specific properties along defined time dimensions. The query interface should provide a list of predefined properties (from T3.1) also a list of predefined time intervals (e.g., last 24 hours, last week, last month)	1
F2	Visualise time-based data as charts. Possible chart types are bar, pie and line charts, or gauge-type visualisations	1
F3	The frontend could be an entry of the last recommendation in the user profile or in the user home page. In this case we plan to show the newest item in the recommendation result list for the active user.	2
F4	Present recommendations as notifications. In this case we could show only the newest item from the recommendation result list that has not been shown yet to the active user.	2
F5	Recommendations could also be available under a specific menu action. When available in this way we could show up to 3 items for each recommendation category.	2

Table 26. Frontend requirements of Prosumer Intelligence Toolkit

#	Requirement	Priority
S1	Store queries and query results as part of the user profile in a database table	1
S2	Store separately the input to the recommendation service (3 stores, one for each of the categories above) and the output of the recommendation service for each user in a database table	2

Table 27. Storage requirements of Prosumer Intelligence Toolkit

5.4.2 System requirements of Möbius Creator's Toolkit

#	Requirement	Priority
B4	Methods to manage opt-in preferences (e.g., allow capture for opted-in users, delete captured properties for opted-out users)	1

B5	Capture, extract and pre-process the requested usage properties (e.g. time a user is reading, clicks, likes, devices used) in real-time as they happen for opted-in users	1
B6	Connection to third party APIs for Text-to-Speech synthesis that includes sending a specified text excerpt and receiving the corresponding audio	3
B7	A method to match text excerpt and audio, aligning text phrase and time-code	2
B8	A method to calculate number of books per publisher, aggregate number of pages and other relevant metrics (for R13)	3
B9	Methods to process contact forms, notify respective actors and store results per user	2
B10	Methods to send chat-like messages and notifications	2
B11	Methods to capture interactions between prosumers/users and publishers	1
B12	Community building methods like connect to a publisher, connect to a user	1
B13	Methods to list established connections	1
B14	Methods to provide autocomplete suggestions for example after three characters are typed in the tags input field	2
B15	Methods to provide the complete list of tags	2
B16	Methods to index tags as part of a full-text and filter-based search index	2
B17	Methods to search and filter based on one or more tags	2
B18	Methods to select allowed categories from a predefined list of feedback categories	3
B19	Methods to store feedback on the specific book item under the respective category	3
B20	Methods to retrieve feedback of a book from the allowed categories	3

B21	Methods to create the appropriate structured metadata for each type of content	3
B22	Methods to render structured metadata in HTML pages	3
B23	Methods to export user data per book and/or author in a CSV format	3
B24	Methods to process, store, transcode and provision audio files in different resolutions	1
B25	Methods to process, store, transcode and provision video files as a stream in different resolutions or as objects to be downloaded	1
B26	Methods to process, store and provision images in different resolutions	1
B27	Methods to process, store and add text or text-based documents	1
B28	Methods to export book related files	1
B29	Methods to search similar text content having as input a text phrase	3
B30	Tools to produce audio narration	1
B31	Tools to produce 3D-audio content including audio effects	1
B32	Methods to store audio narration, 3D-audio content, and other multimedia files	1
B33	Methods to align audio narration and 3D audio content with text segments	1

Table 28. Backend requirements of Möbius Creator's Toolkit

#	Requirement	Priority
F6	Opt-in form per user, as part of the user registration, change of opt-in as part of a user preference page (for R10, R11, R14)	1
F7	Construct a query so that users can request specific properties along defined time dimensions. The query interface should provide a list of predefined properties (e.g., time read, clicks. devices) also a list of predefined time intervals (e.g., last 24 hours, last week, last month)	1

F8	Visualise time-based data as charts. Possible chart types are bar, pie and line charts, or gauge-type visualisations	1
F9	An extended audio player able to play the audio time-codes and highlight the corresponding text in a synchronised manner	3
F10	An audio player able to play the audio-time codes for use-cases related to audio-books only	3
F11	An entry in a dashboard showing the corresponding metrics (for R13)	3
F12	Interface to send and receive chat-like messages together with missing notifications	2
F13	Fill contact form, validate form fields	2
F14	Input form to search for books which hold one, or more tags. The input form could provide autocomplete suggestions.	2
F15	Input form to add one, or more tags to a book. The input form could provide autocomplete suggestions.	2
F16	A component to add feedback that includes a emoji-style reaction and an optional text field	3
F17	A component to show add feedback and the category it belongs	3
F18	Interface to add any structured metadata that are not part of the book/author data model	3
F19	Interface to request the CSV export	3
F20	Interface to upload files, add file descriptions and tags	1
F21	Interface to manage (e.g., edit/delete) and to link files to specific segments of books	1
F22	Interface to export all book related files in an EPUB container, which essentially is a single unencrypted zipped archive containing all book resources	1
F23	Interface to highlight a text segment (possibly limited to a maximum number of characters) and initiate a similarity search	3
F24	Interface to show the results of the similarity search	3

F25	Interface to produce audio narration (start/stop recording)	1
F26	Interface to aid the production of 3D content, e.g., for setting the sound field and sound directions	1
F27	Interface to upload audio narration, 3D-audio, and other multimedia files	1
F28	Interface to align 3D-audio effects, photos, videos to specific text segments	1
F29	Interface to show the existence of available additional information	1

Table 29. Frontend requirements of Möbius Creator's Toolkit

#	Requirement	Priority
S3	Store the requested properties as part of the user profile and additionally in an anonymised aggregated form as time-series data in a database table.	1
S4	Depending on the number of concurrent users and interactions, a heavy load can be placed on the database. In order to mitigate this risk, we could use an in-memory key store (e.g., memcached or redis) as a temporary cache.	2
S5	A fast read/write file storage for the corresponding audio files that allows for adequate provision of bandwidth (e.g., to cater for concurrent users requesting audio files)	3
S6	Store identifiers to text synthesized and corresponding identifiers to audio files for each book title	3
S7	Store for each publisher the identifiers of each book and its number of pages in a database table	2
S8	Store messages as part of the user interactions in a database table together with their delivery status (e.g., sent, pending) and message modality (e.g., contact form, chat)	2
S9	Store contact details and opt-in preferences of prosumers, users and publishers in a database table	2

S10	Extension of the book data model with an additional property “tags” property. Tags could be a list of terms that are shared among users.	2
S11	Store feedback properties (category reaction, text) referencing the book identifier and the user identifier in a database table	3
S12	Store predefined categories of feedback in a database table	3
S13	Store predefined reactions for feedback in a database table	3
S14	Store additional structured metadata, that are not captured previously as part of the book/author data model	3
S15	File storage of CSV exports with additional downloading facilities.	3
S16	Store identifiers to exports and file locations in the database table	3
S17	A fast IO file storage, potentially under the same book folder	1
S18	Store identifiers of files, their locations, other file metadata along with book identifiers in a database table.	1
S19	Store additional search indexes in Apache Solr	3
S20	File storage for all assets (see also R20) as part of a book entity	1
S21	Store asset identifiers and related book identifiers in a database table	1

Table 30. Storage requirements of Möbius Creator’s Toolkit

5.4.3 System requirements of Möbius Book Player

#	Requirement	Priority
B30	Methods to add forum entries, possibly with an upvoting feature and insertion of tags and uploading of photos, and audio-visual files	2
B31	Methods to index forum entries as part of the search index	2
B32	Methods to search based on full-text, filter based on tags	2
B33	Methods to mark discussions as active or stale based on recency and number of entries in a predefined time period.	2

B34	Methods to aggregate tags and provide means to explore content, e.g., using a tag cloud visualisation	2
B35	Methods to add replies and organise discussions in threads	2
B36	Methods to create a dedicated hashed link for a book. Users with the link can comment and review but not edit content	3
B37	Methods to invite reviewers for providing comments and suggestions	3
B38	Methods to attach comments to specific text segments	3
B39	Methods to accept comments and suggestions	3
B40	Methods to connect text segments of a book with external links	2
B41	Methods to automatically capture reading state, possibly on a page level if only the text modality is selected, and to the segment level is a combined text/audio modality is selected	1
B42	Methods to list all assets of a book, potentially groups by type (e.g., photo, video)	1
B43	Methods to calculate a content quality score based on user-ratings and feedback	1
B44	Methods to sort by quality score	1

Table 31. Backend requirements of Möbius Book Player

#	Requirement	Priority
F30	Interface to add forum entries	2
F31	Interface to show active discussions	2
F32	Interface to search discussions based on tags and display results	2
F33	Interface to show threaded discussions	2
F34	Interface for reviewers to select a specific text segment and add comment	3
F35	Interface for authors to accept comments	3

F36	Interface to show the existence of these additional links	2
F37	Interface to manually capture the reading state e.g., “Continue on the audio mode”	1
F38	Interface to initiate the display of all available assets when a user opens additional information associated to a text segment	1
F39	Interface to display all available assets when a user requests it	1
F40	Interface to show the sorted results (content by quality score)	1

Table 32. Frontend requirements of Möbius Book Player

#	Requirement	Priority
S22	Store forum entry identifiers, text, tags, file identifiers, replies and thread identifiers in a database table	2
S23	Store file identifiers, entry identifiers and file locations in a database table	1
S24	File storage for related photos and audio-visual files	1
S25	Store book identifier, user identifiers and their comments on specific text segments and the current status of the comment (e.g., accepted, rejected) in a database table	3
S26	Store link identifiers and link locations on text segments together their book identifiers in a database table	2
S27	Store text segment, page of the current reading state for each user and book in a database table	1
S28	Store quality score and its evolution across time for each book in a database table	1

Table 33. Storage requirements of Möbius Book Player

5.5 Additional functional requirements for the container format

In order to determine the functional requirements for the container format we considered the four scenarios in which the Möbius Book will be experienced.

5.5.1 Scenario 1: single reader, single (personal) screen, variable layout

This is the basic "multimodal eBook" use case, where a single user can choose between different modes of reading, listening and viewing and create their own multimodal mix, enabling and disabling the different modes as they please, modify font size and appearance, and select the narration speed, etc, similar to a typical eBook / audiobook.

Content types include:

- Text
- Narration
- Sound effects (3D audio)
- Music (3D audio)
- Video segments
- Static images

Text is arranged in a traditional book style (it's not flying over the screen or animated).

Text is reflowable to adapt to screen- and font size, similar to a classic eBook reader.

Audio is reproduced in binaural stereo, targeting headphones.

If all multimodal content is disabled, then it isn't any different from a regular eBook. If narration or audio content is activated, the user will be "guided" through the text by highlighting the currently narrated word or sentence.

5.5.2 Scenario 2: single reader, single (personal) screen, fixed layout

Similar to scenario one, except that the user has less freedom in modifying or selecting layout and appearance.

Text layout is fixed and the multimodal content can't be disabled. This is the "individual transmedia storytelling" use case. The user is guided in their reading by highlighted text passages.

5.5.3 Scenario 3: one or multiple readers, single (public) screens, fixed layout

This is the basic installation scenario, where the transmedia stories are presented in an exhibition setting, on one big screen. The layout and multimodal content reproduced at a fixed reading speed. Audio can either be reproduced over headphones or a loudspeaker setup (mono, stereo, 3D). The readers will be "guided" through the text by highlighting the current word or sentence. Basically, use case 2, but with a bigger screen.

5.5.4 Scenario 4: one or multiple readers, multiple (public) screens, fixed

layout

Same as the previous use case, except that the content is distributed over multiple screens. Audio can be reproduced over headphones or a loudspeaker setup (mono, stereo, 3D). The readers will be "guided" through the text by highlighting the current word or sentence.

5.5.5 Additional functional requirements

#	Requirement
FR1	Needs to specify the content format (fixed layout, variable layout) with a higher granularity (i.e., which content can be disabled, which content can't be disabled).
FR2	Needs to be able to hold ambisonic audio files that can be reproduced on a variety of target setups. This requires the format to hold uncompressed audio (currently only mp4 and mp3 are allowed by the standard).[53]
FR3	Needs to be able to hold video files that can be reproduced on a variety of target setups.
FR4	Need to be able to associate multiple audio segments with a text passage (currently only one audio segment is allowed)
FR5	Needs to be able to associate multiple video segments with a text passage (currently video can only be embedded in the content document).
FR6	Needs to be able to associate visual content (text passages, images, video) with a prospective output screen.
FR7	Needs to be able to display reflowable text on various screen sizes.
FR8	Needs to be able to display fixed-layout text on various screen sizes.
FR9	Needs to be able to play back (spatial) voice narration of the text.
FR10	Needs to be able to highlight the text in synchronization to the narration being read.
FR11	Needs to be able to scroll the pages synchronous to the voice narration.
FR12	Needs to be able to play back a (3D) sound effects track synchronous to the narration.

[53]Note that by using ambisonic audio, we can avoid having to specify individual positions for sound sources, as these are implicit in the format. Furthermore, the format can be rendered to various output configurations.

FR13	Needs to be able to play back a (3D) music track synchronous to the narration.
FR14	Needs to be able to play back videos (synchronous to the narration or when landing on a certain text passage).
FR15	Needs to be able to reproduce ambisonic audio material that can be reproduced on a variety of target setups.
FR16	Needs to be able to display images once the reader reaches a certain text passage.
FR17	Needs to incorporate "classic" eBook features, such as a "bookshelf" view, page navigation, etc.
FR18	Needs to provide controls over the multimodal content, i.e. to enable or disable the respective audio tracks, video content, and so on, if the container file allows it.
FR19	Needs to incorporate "transport" features in case a user selects guided narration (in which case page turning will be automatic, etc), similar to an audio player (Play, Pause, Stop, Fast Forward, etc).
FR20	Needs to provide a configuration interface for screen- and audio setup.

Table 34. Functional requirements for the container format

6 System architecture and user workflow

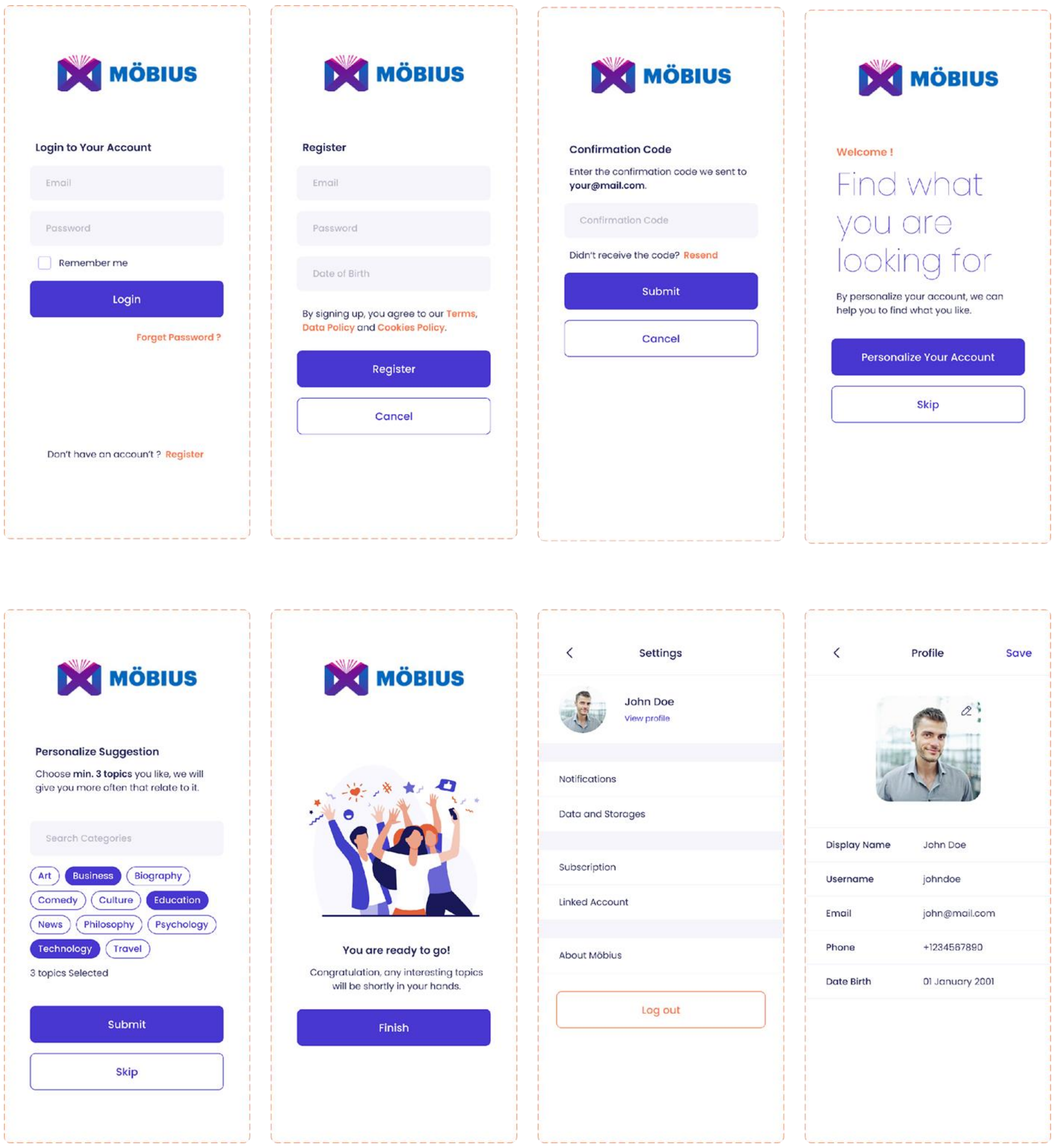
After having considered the different components of the Möbius system that were introduced by the user requirements, this chapter deals with the development of architectural diagrams for each of the Möbius envisioned outcomes.

We design the Möbius system following the principles of a service-based architecture. This has significant advantages with regards to other architecture types (e.g., monolithic or layered-based architecture) including better scalability, easier testability of components, faster deployment of services without any downtimes. Furthermore, this architectural approach gives the opportunity to design self-contained services which in turn help to better manage and control them and are easier to maintain. Generally, service-based architectures are also designed to be more flexible and modular, using loosely coupled components. One major advantage of this approach is that individual components can be improved and even replaced, independently and don't require large management efforts between teams of developers, as long as the service contracts (i.e., the agreed APIs[54] between the services) are maintained. In Möbius we implement APIs using REST[55] design principles.

In service-based architectures, there is a trade-off that has to be made with regards to the extent of the functionality a service should cover. There are cases where very granular services, a term that is known as microservices, make sense but they tend to increase complexity in large systems as Möbius. In Möbius we opted to package services in terms of functional components, following the user requirements and the elaboration of functional requirements presented above.

Since the user registration / authentication workflow is common for all Möbius outcomes, we created the following user flow. The aim is to avoid unnecessarily complications in the architectural diagrams. The user flow is presented on a mobile device primarily from a reader perspective, it is planned to be the same though also on the web. It is worth to be noted that in addition to readers, Möbius users could be publishers and prosumers. Their user authentication flow is similar to the one shown below, readers, publisher, prosumers however have access to different parts of the Möbius system (see section below).

[54]What is an Application Programming Interface (API), https://www.ibm.com/cloud/learn/api
[55]REST APIs, https://www.ibm.com/cloud/learn/rest-apis



MÖBIUS

Login to Your Account

Email

Password

Remember me

Login

Forgot Password ?

Don't have an account ? Register

MÖBIUS

Register

Email

Password

Date of Birth

By signing up, you agree to our Terms, Data Policy and Cookies Policy.

Register

Cancel

MÖBIUS

Confirmation Code

Enter the confirmation code we sent to your@mail.com.

Confirmation Code

Didn't receive the code? Resend

Submit

Cancel

MÖBIUS

Welcome !

Find what you are looking for

By personalize your account, we can help you to find what you like.

Personalize Your Account

Skip

MÖBIUS

Personalize Suggestion

Choose min. 3 topics you like, we will give you more often that relate to it.

Search Categories

Art · Business · Biography · Comedy · Culture · Education · News · Philosophy · Psychology · Technology · Travel

3 topics Selected

Submit

Skip

MÖBIUS

You are ready to go!

Congratulation, any interesting topics will be shortly in your hands.

Finish

Settings

John Doe

View profile

Notifications

Data and Storages

Subscription

Linked Account

About Möbius

Log out

Profile | Save

Display Name	John Doe
Username	johndoe
Email	john@mail.com
Phone	+1234567890
Date Birth	01 January 2001

Figure 2. User registration and authentication workflow

6.1 Prosumer Intelligence Toolkit

In this section we synthesize the requirements presented above and give an overview of the Prosumer Intelligence Toolkit, as well its main functionality. From a user perspective, the interface to this toolkit is primarily a dashboard where queries can be formulated and the corresponding results are presented. The component for knowledge extraction from the fanfiction communities thus provides an API to reflect the user query parameters and filter results based on their requests.

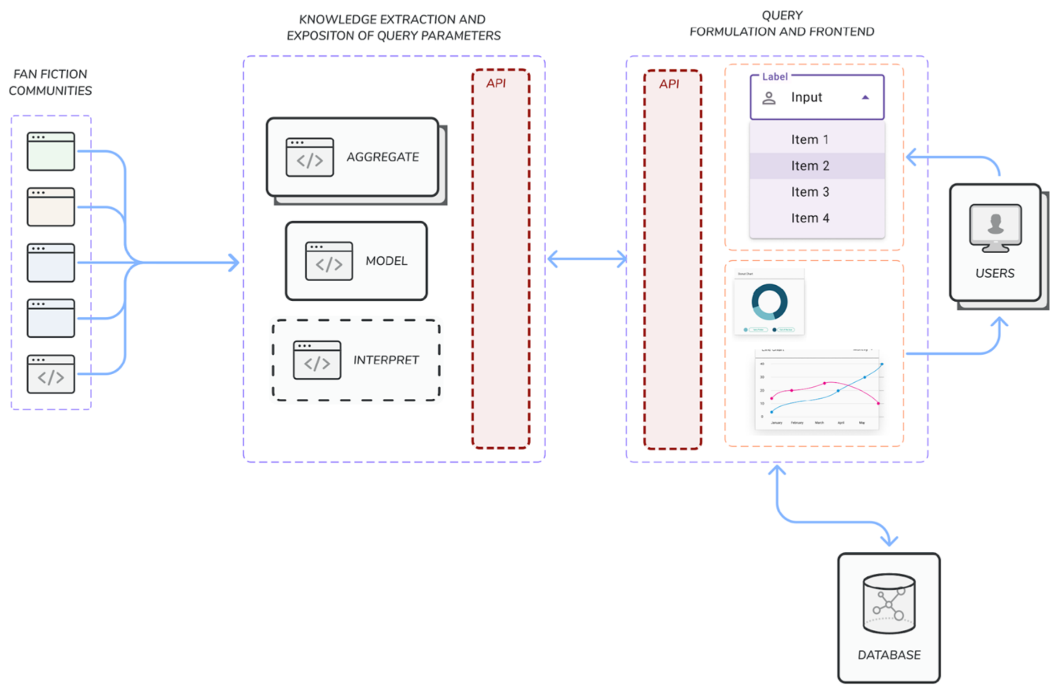


Figure 3.Architecture Prosumer Intelligence Toolkit

6.2 Möbius Book

In this section we will present the architecture for the Möbius Book Creator's Toolkit and the Möbius Player. Both Möbius outcomes have access to the same components, thus in the Möbius Book Creator's Toolkit we will present how these components interconnect and in the Möbius Book Player a possible user interface for readers.

6.2.1 Möbius Book Creator's Toolkit

The figure below shows the overall architecture of the Möbius Book Creatot's toolkit. In order to make this schematic diagram clearer, the next figures show sections of this diagram as a way to zoom in to specific parts. Please note the legend on the top-left or bottom-right corner that shows which specific part of the architecture is currently displayed. For the main user-facing components of the architecture, we also present samples of the user interface to provide impressions of the look and feel.

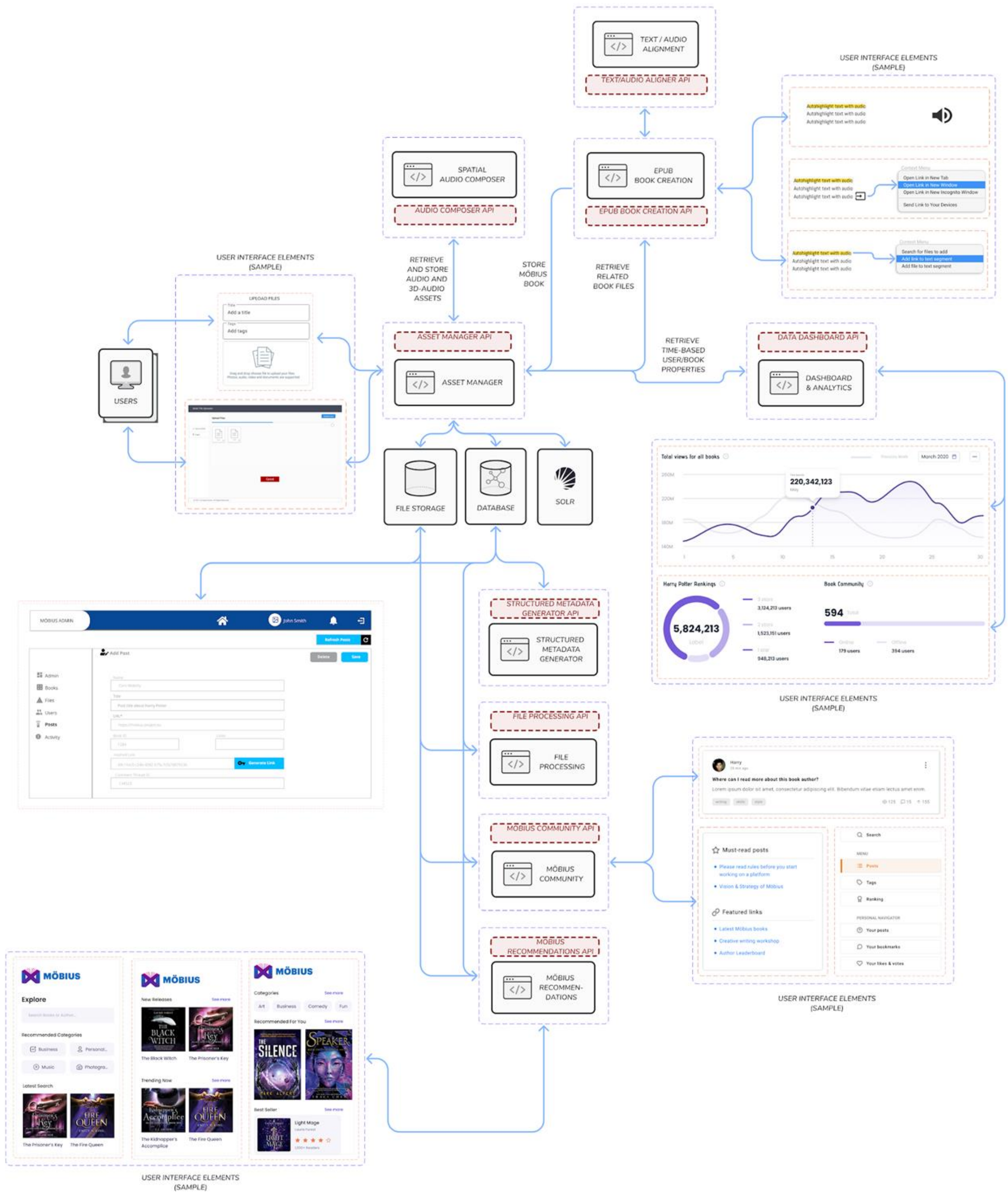

TEXT / AUDIO ALIGNMENT
TEXT/AUDIO ALIGNER API
USER INTERFACE ELEMENTS (SAMPLE)
SPATIAL AUDIO COMPOSER
AUDIO COMPOSER API
EPUB BOOK CREATION
EPUB BOOK CREATION API
USER INTERFACE ELEMENTS (SAMPLE)
RETRIEVE AND STORE AUDIO AND 3D-AUDIO ASSETS
STORE MÖBIUS BOOK
RETRIEVE RELATED BOOK FILES
USERS
ASSET MANAGER API
ASSET MANAGER
RETRIEVE TIME-BASED USER/BOOK PROPERTIES
DATA DASHBOARD API
DASHBOARD & ANALYTICS
FILE STORAGE
DATABASE
SOLR
STRUCTURED METADATA GENERATOR API
STRUCTURED METADATA GENERATOR
USER INTERFACE ELEMENTS (SAMPLE)
FILE PROCESSING API
FILE PROCESSING
MÖBIUS COMMUNITY API
MÖBIUS COMMUNITY
MÖBIUS RECOMMENDATIONS API
MÖBIUS RECOMMEN-DATIONS
USER INTERFACE ELEMENTS (SAMPLE)
USER INTERFACE ELEMENTS (SAMPLE)

Figure 4. Overall Architecture of the Möbius Creators' Toolkit

The figure below shows the relevant Möbius architecture parts that deal with asset management, and 3D audio production, as well an interface to upload and view assets in Möbius

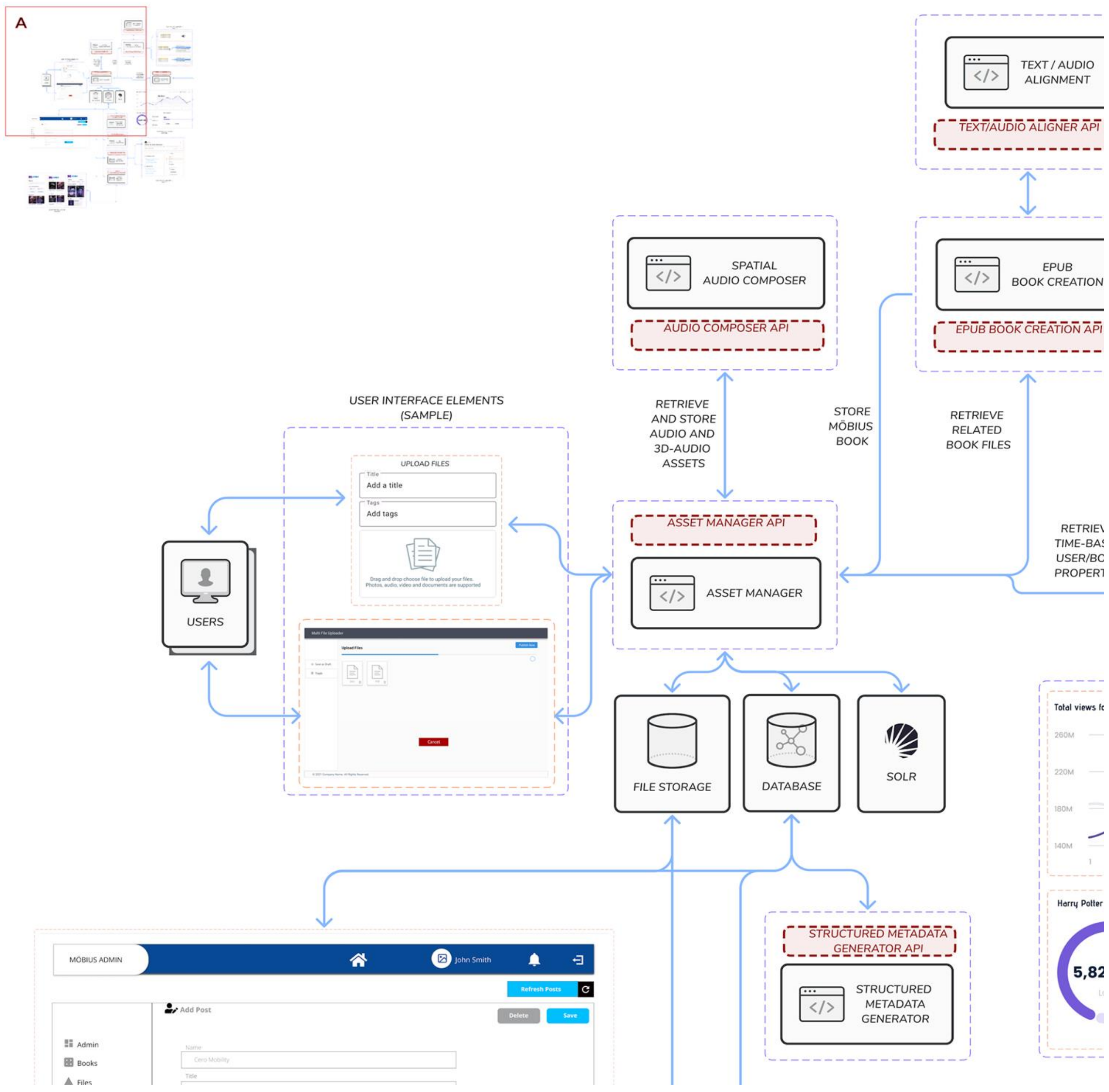


Figure 5. Möbius Creator's Toolkit Architecture – Part A

The figure below presents the part of the Möbius architecture that deal with the creation of eBooks, the capturing and presentation of analytics as well the structured metadata creation component.

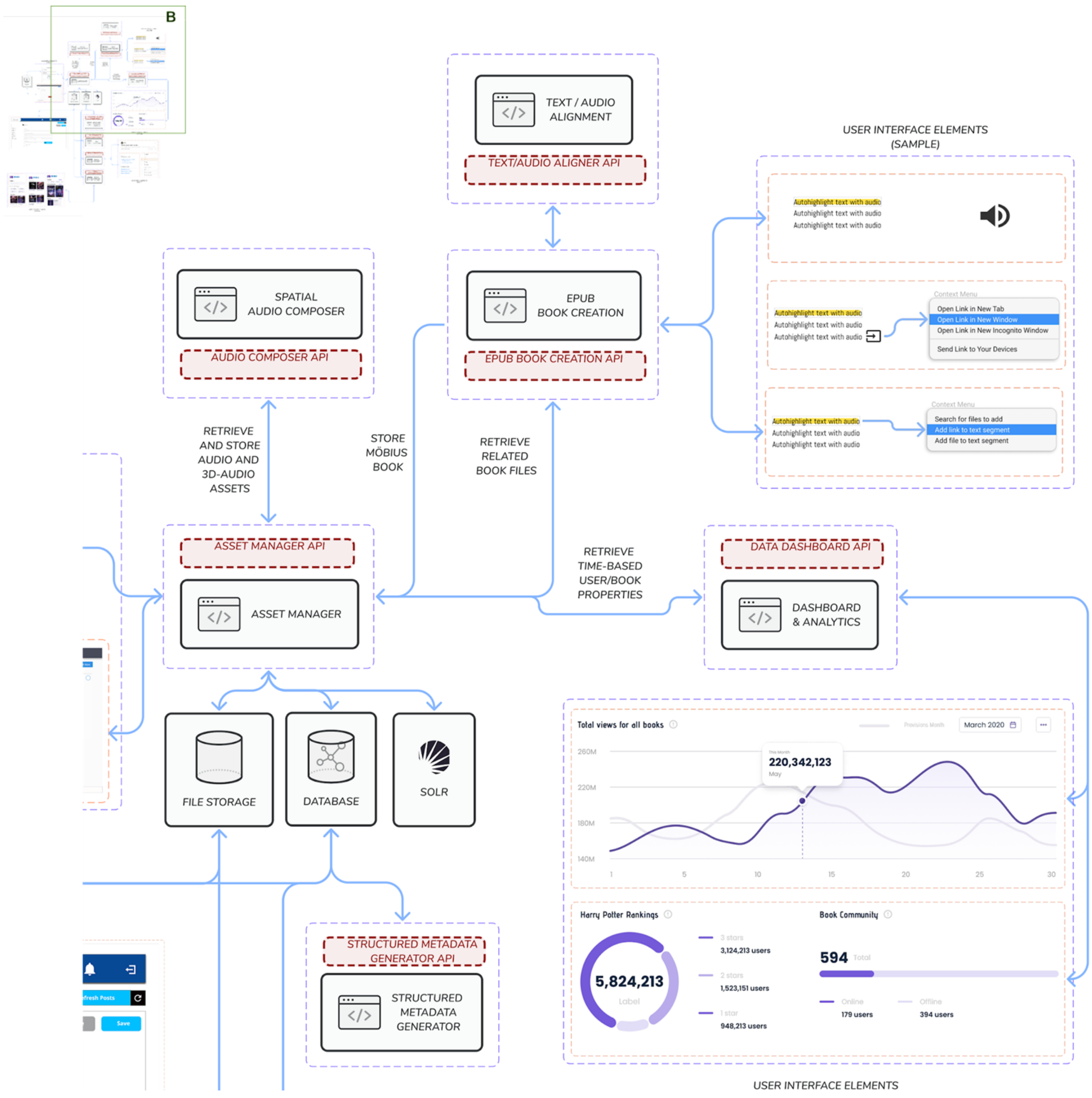


Figure 6. Möbius Creator's Toolkit Architecture – Part B

The figure below provides an interface for an administrator screen together with components that deal with file processing and recommendations. Additionally, we also show how recommendations could be visualised In reader screens.

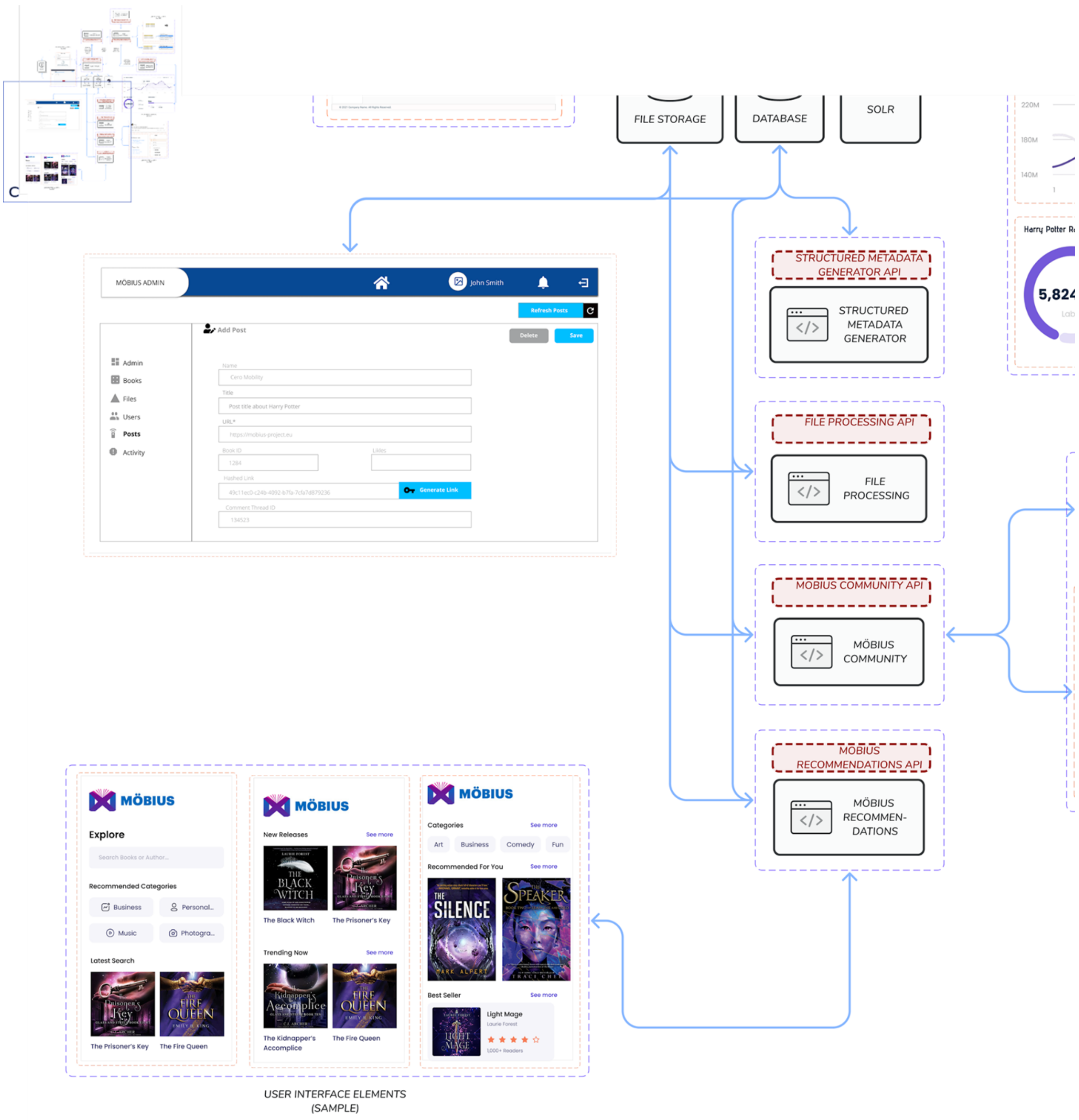
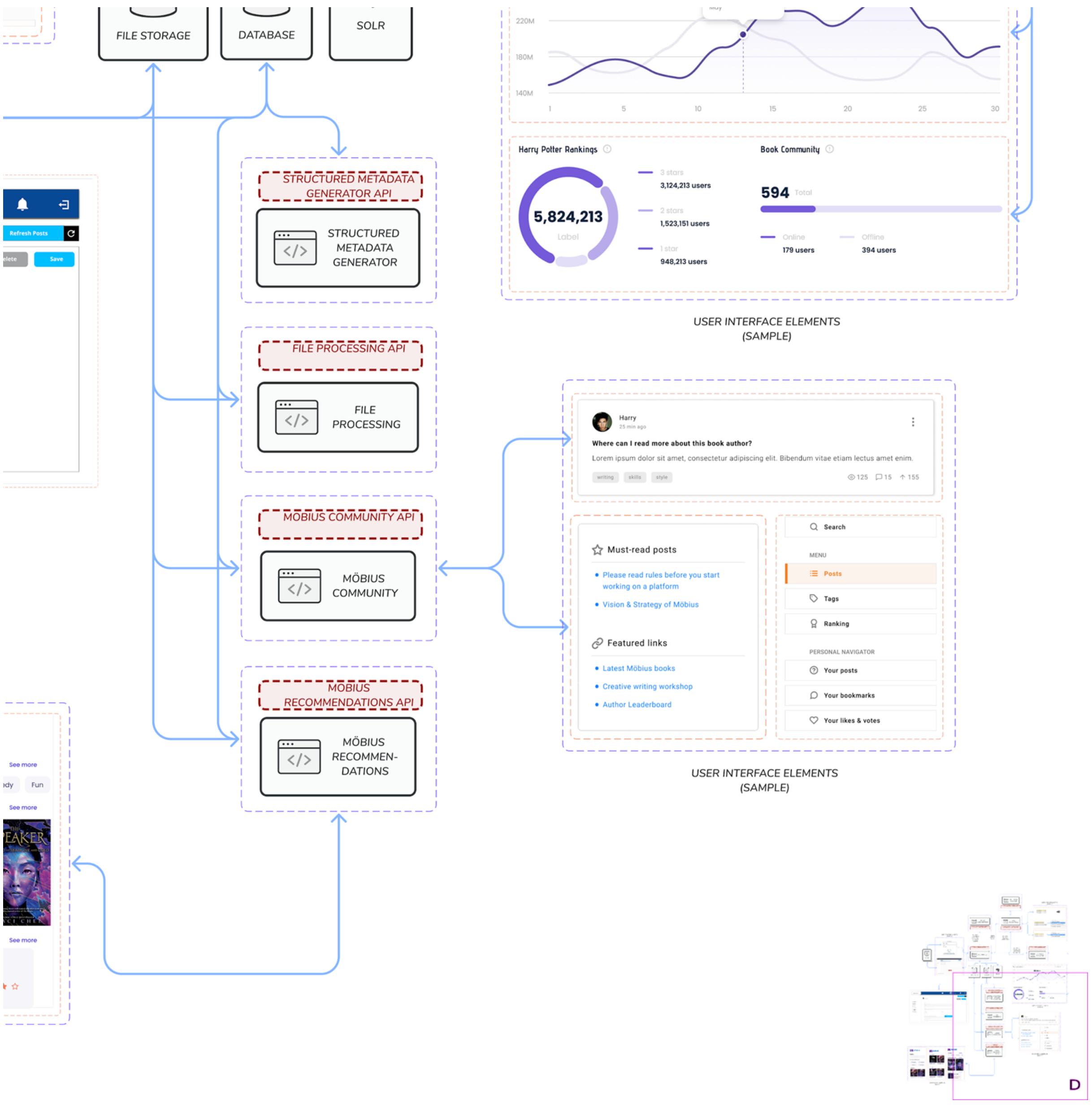


Figure 7. Möbius Creator's Toolkit Architecture – Part C

The figure below shows the Möbius community and a possible interface to add posts, tags and explore content.



FILE STORAGE
DATABASE
SOLR
STRUCTURED METADATA GENERATOR API
STRUCTURED METADATA GENERATOR
FILE PROCESSING API
FILE PROCESSING
MÖBIUS COMMUNITY API
MÖBIUS COMMUNITY
MÖBIUS RECOMMENDATIONS API
MÖBIUS RECOMMEN-DATIONS
USER INTERFACE ELEMENTS (SAMPLE)
USER INTERFACE ELEMENTS (SAMPLE)
D

Figure 8. Möbius Creator's Toolkit Architecture – Part D

6.2.2 Möbius Book Player

On the frontend, the Möbius Creators Toolkit will be implemented as a modern web-application, while the Möbius player, which will be primarily used on mobile devices, will be implemented as a progressive application[56], using common web technologies (HTML5, JavaScript, CSS) as this provides a number of advantages such as excellent portability regardless of the end-user's hardware or device operating system. For end-users the Player

[56]Progressive Web Application, https://en.wikipedia.org/wiki/Progressive_web_application

can be treated as any other native application (e.g., working offline) with the added advantage that it does not require any complex installation or periodic updates.

Since the Möbius player and the Möbius Creators' Toolkit should have access to the same data (e.g., writers, books, community content) we aim to design the Möbius player on top of selected services presented above. In order to satisfy the user requirements, the Möbius player should use the Media Asset Management, the Möbius recommendations and the Möbius community for example to allow readers to download Möbius books, receive recommendations and get in touch with writers.

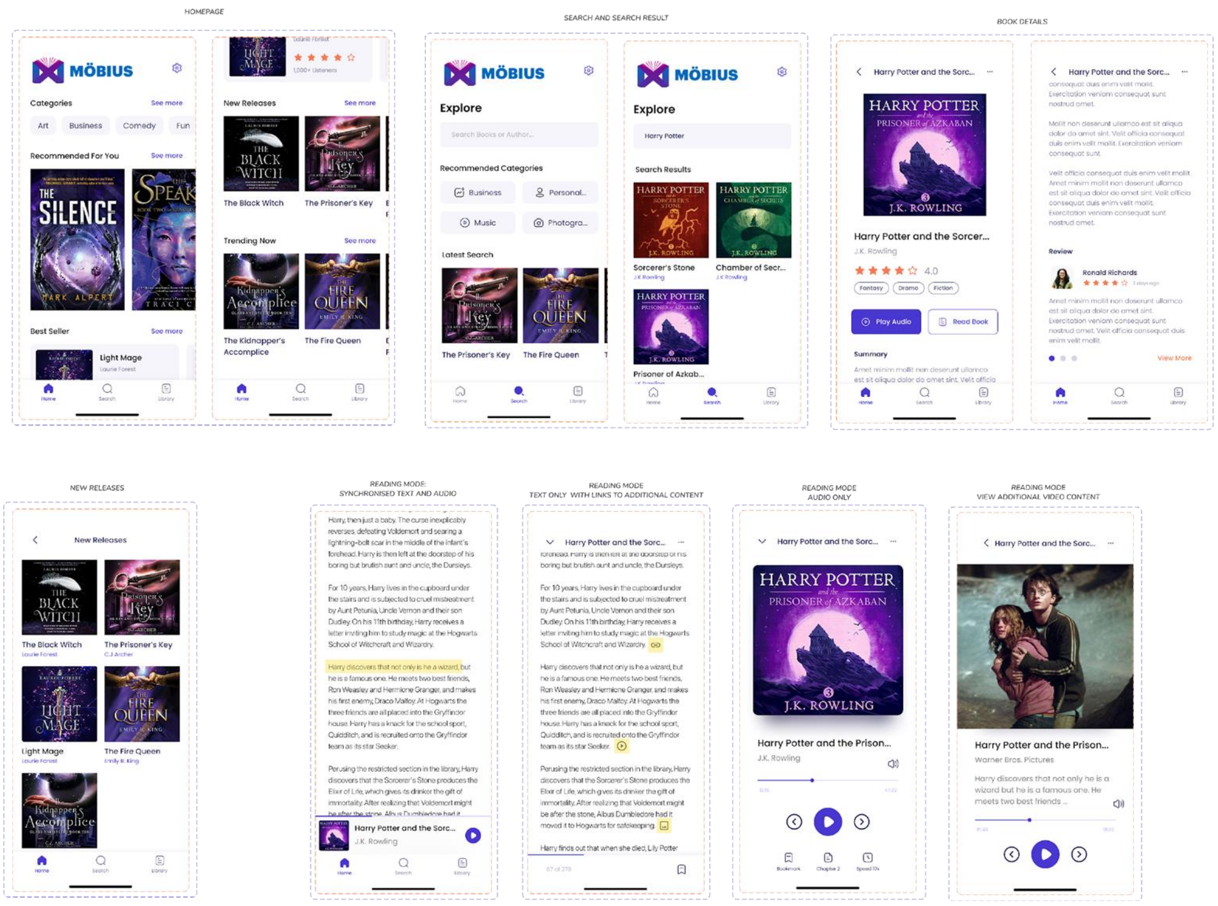


Figure 9. Möbius Book Player – Reader Interface

6.2.3 User roles

From the user requirements and the elaboration of functional requirements above, we can see that the user types in Möbius have access to different parts of the Möbius tools. We plan to manage this using a concept that is called user roles. User roles are essentially permissions that are set or granted and allow access to areas and features of the Möbius system. They allow a flexible update of permissions and also the addition of new roles as needed. The next table shows the current assignment of Möbius user roles and access permissions.

User Role	Prosumer Intelligence Toolkit	Möbius Creators Toolkit	Möbius Book Player
Reader	No access	Partial access (Community, Recommendations, Asset Manager)	Full access
Prosumer	Full access	Full Access	Full Access (if needed)
Publisher	Full Access	Partial Access (Community, Recommendations, Asset Manager)	Full Access (if needed)
Administrator	Full access	Full access	Full access

Table 35. User roles in Möbius

6.3 API description

This section builds on top of the functional requirements and the components identified to create the first set of the current Möbius data models, methods and API descriptions. We follow a simplified JAVA notation[57] that describes classes, objects, object properties as well methods to manipulate and construct objects within a class. Furthermore, we use annotations[58] to denote either how the data element will be presented on a JSON output (e.g., @SerializedName) or the relations between data models (e.g., @OneToOne). The table below shows the current data models and methods that need to implemented for the Möbius envisioned outcomes.

[57]Classes and Objects, https://docs.oracle.com/javase/tutorial/java/javaOO/index.html
[58]Annotation Basics, https://docs.oracle.com/javase/tutorial/java/annotations/basics.html

Data models and related methods

```
public class FanFictionCommunity {
    @SerializedName("url")
    String url;
    @SerializedName("token")
    String token;
    @SerializedName("actors")
    List<Actors> actors;
    @SerializedName("entries")
    List<Entry> entries;
    @SerializedName("interactions")
    List<Interaction> interactions;

    public List<Interaction> getInteractions(Long actorId);
    public List<Entry> getEntries(Long actorId);
    public Actor getActorByEntry(Long entryId);
    public Actor getActorByInteraction(Long interactionId);
}
```

```
public class Entry {
    @SerializedName("title")
    String title;
    @SerializedName("description")
    String description;
    @OneToMany
    @SerializedName("book")
    Book book;
    @ManyToOne
    @SerializedName("actor")
    Actor actor;
    @SerializedName("comments")
    List<Comment> comments;
    @SerializedName("reactions")
    List<Reaction> reactions;
    @SerializedName("pubDate")
    Date pubDate;

    public List<Comment> getComments(Long actorId, Long bookId);
    public List<Reaction> getReactions(Long actorId, Long bookId);

    public Entry(String title, String description, Long bookId, Long actorId);
}
```

Data models and related methods

`public class Actor {`
`    Enum Type {`
`        READER,`
`        PROSUMER,`
`        AUTHOR,`
`        PUBLISHER,`
`        ADMIN;`
`    }`
`    @SerializedName("username")`
`    String username;`
`    @SerializedName("email")`
`    String email;`
`    @SerializedName("type")`
`    Type type;`
`    @OneToOne`
`    @SerializedName("profile")`
`    Profile profile;`
 	
`    public Actor(String username, String email, Type type);`
`}`

`public class Comment {`
`    @SerializedName("text")`
`    String text;`
`    @SerializedName("replyToCommentId")`
`    Long replyToCommentId;`
`    @ManyToOne`
`    @SerializedName("entry")`
`    Entry entry;`
`    @SerializedName("pubDate")`
`    Date pubDate;`
 	
`    public List<Comment> getReplies(Long commentId);`
 	
`    public Comment(String text, Long replytoCommentID, Long entryId);`
`}`

Data models and related methods
```
public class Reaction {
    Enum Type {
        LIKE,
        SHARE,
        COMMENT;
    }
    @SerializedName("type")
    Type type;
    @ManyToOne
    @SerializedName("actor")
    Actor actor;
    @ManyToOne
    @SerializedName("entry")
    Entry entry;
    @ManyToOne
    @SerializedName("comment")
    Comment comment;
    @ManyToOne
    @SerializedName("book")
    Book book;

    public List<Entry> getEntries(Long actorId);
    public List<Comment> getComments(Long actorId);
    public List<Reaction> getReactions(Long actorId);
    public List<Reaction> getReactionsByBook(Long bookId);
    public List<Reaction> getReactionsByActorAndBook(Long actorId, Long bookId);

    public Reaction(Long actorId, Long bookId, Type type);
}
```
```
public class Profile {
    @SerializedName("firstName")
    String firstName;
    @SerializedName("lastName")
    String lastName;
    @SerializedName("bio")
    String bio;
    @SerializedName("consent")
    Boolean consent = false;
    @OneToOne
    @SerializedName("actor")
    Actor actor;

    public Profile(String firstName, String lastName, String bio, Boolean consent, Long actorId);
}
```

Data models and related methods

```
public class Interaction {
    Enum Action {
        READING,
        WRITING,
        LIKE,
        SHARE,
        CLICK;
    }
    Enum Device {
        MOBILE,
        TABLET,
        DESKTOP;
    }
    @SerializedName("startTime")
    Date startTime;
    @SerializedName("endTime")
    Date endTime;
    @SerializedName("action")
    Action action;
    @SerializedName("device")
    Device device;
    @ManyToOne;
    @SerializedName("book")
    Book book;
    @OneToMany
    @SerializedName("actor")
    Actor actor;

    public List<Action> getActions(Long bookId);
    public List<Action> getActionsByActor(Long actorId);
    public List<Device> getDevices(Long bookId);
    public List<Device> getDevicesByActor(Long actorId);

    public Interaction(Device device, Action action, Long bookId, Long actorId);
}
```

Data models and related methods
```
public class Book {
    @SerializedName("author")
    Author author;
    @SerializedName("publisher")
    Publisher publisher;
    @SerializedName("metadata")
    Metadata metadata;
    @SerializedName("package")
    Package package;
    @SerializedName("bookThumb")
    String bookThumb;

    public Package getPackage(Long bookId);
    public Publisher getPublisher(Long bookId);
    public Metadata getMetadata(Long bookId);

    public Book(Long authorId, Long publisherId, String bookThumb);
}
```
```
public class Author {
    @SerializedName("email")
    String email;
    @SerializedName("phone")
    String phone;
    @SerializedName("profile")
    Profile profile;

    public Author(String email, String phone, Long profileId);
}
```
```
public class Publisher {
    @SerializedName("email")
    String email;
    @SerializedName("phone")
    String phone;
    @SerializedName("profile")
    Profile profile;

    public Publisher(String email, String phone, Long profileId);
}
```

Data models and related methods

```
public class Package {
    @SerializedName("path")
    String path;
    @SerializedName("hash")
    String hash;
    @SerializedName("packageName")
    String packageName;
    @SerializedName("assets")
    List<Asset> assets;
    @OneToMany
    @SerializedName("book")
    Book book;

    public List<Asset> getAssets(Long bookId);

    public Package(String path, String packageName, Long bookId);
}
```

```
public class Metadata {
    @SerializedName("isbn")
    String isbn;
    @SerializedName("type")
    String type;
    @SerializedName("product")
    Product product;
    @SerializedName("extent")
    Extent extent;

public Metadata(String isbn, String type, Long productId, Long extentId);
}
```

```
public class Extent {
    @SerializedName("type")
    String type;
    @SerializedName("value")
    String value;
    @SerializedName("unit")
    String unit;

    public Extent(String type, String value, String unit);
}
```

Data models and related methods
```
public class Product {
    @SerializedName("form")
    String form;
    @SerializedName("formType")
    String formType;
    @SerializedName("feature")
    String feature;

    public Product(String form, String formType, String feature);
}
```
```
public class User extends Actor, Publisher {
    @Transient
    String plainTextPassword;
    String password;
    @OneToOne
    @SerializedName("actor")
    Actor actor;
    @OneToMany
    @SerializedName("publisher")
    Publisher publisher

    public User registerUser(Long userId, String plainTextPassword);
    public User(String plainTextPassword);
}
```

Data models and related methods

```
public class Asset {
    Enum Type {
        FILE,
        LINK;
    }
    @SerializedName("type")
    Type type;
    @SerializedName("path")
    String path;
    @SerializedName("fileName")
    String fileName;
    @SerializedName("hash")
    String hash;
    @SerializedName("MöbiusFile")
    MöbiusFile MöbiusFile;
    @SerializedName("MöbiusLink")
    MöbiusLink MöbiusLink;
    @SerializedName("segment")
    Segment segment;

    public Asset getAssetBySegment(Long segmentId);

    public Asset(String path, String fileName, Long segmentId);
}
```

```
public class MöbiusFile {
    Enum Format {
        AUDIO,
        AUDIO_EFFECTS,
        3D_AUDIO,
        PHOTO,
        VIDEO;
    }
    @SerializedName("path")
    String path;
    @SerializedName("fileName")
    String fileName;
    @SerializedName("hash")
    String hash;
    @SerializedName("format")
    Format format;

    public MöbiusFile(String path, String fileName, Format format);
}
```

Data models and related methods
```
public class MöbiusLink {
    @SerializedName("url")
    String url;
    @SerializedName("title")
    String title;

    public MöbiusLink(String url, String title);
}
```
```
public class Segment {
    Enum Type {
        TIMECODE,
        TEXT;
    }
    @SerializedName("type")
    Type type;
    @SerializedName("start")
    Long start;
    @SerializedName("end")
    Long end;
    @OneToMany
    @SerializedName("book")
    Book book

    public List<Segment> getSegments(Long bookId);

    public Segment(Type type, Long start, Long end, Long bookId);
}
```
```
public class MöbiusCommunity {
    @SerializedName("users")
    List<User> users;
    @SerializedName("comments")
    List<Comment> comments;
    @SerializedName("reactions")
    List<Reaction> reactions;
    @SerializedName("books")
    List<Book> books;

    public List<Book> getBooksByReaction(Long reactionId);
    public List<Book> getBooksByUser(Long userId);
    public Book getBookByComment(Long commentId);
}
```

Data models and related methods

```
public class Recommendation {
    @ManyToOne
    @SerializedName("user")
    User user;
    @SerializedName("authors")
    List<Author> authors;
    @SerializedName("books")
    List<Book> books;

    public List<Author> getAuthorRecommendations(Long userId);
    public List<Book> getBookRecommendations(Long userId);

    public Recommendation(Long userId, Long authorId, Long bookId)
}
```

Table 36. Möbius annotated data models and methods

From the above descriptions it is straight-forward to to generate API definitions. Below we show how a sample JSON[59] response will be generated by going through an example. In the Model-View-Controller paradigm for web-based applications, methods (or also called actions) are triggered by routes. Routes represent URL endpoints where HTTP requests are being sent and are essentially the part of the URL without the host domain. We make the following assumptions:

- The route `/get/author/recommendations/{{userId}}` triggers the method `getAuthorRecommendations(Long userId)` from the `Recommendation` class above.
- The `userId` with a value `123456789` points to a valid and existing user in the database
- The `getAuthorRecommendations(Long userId)` method returns up to the 3 results sorted by relevance.

Issuing a HTTP GET request[60] then to `/get/author/recommendations/123456789` will execute the corresponding method, the result of which is a List<Author>, that is serialised with the name “authors“. Because the result is a list of authors, it essentially converts the output into a JSON array. As we see below an Author has an email, phone and profile property that are serialised by ”email”, ”phone” and ”profile” accordingly.

[59]JSON (JavaScript Object Notation, https://en.wikipedia.org/wiki/JSON
[60]A HTTP GET request is effectively the same as opening a URL in the browser, see more https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

```
public class Author {
    @SerializedName("email")
    String email;
    @SerializedName("phone")
    String phone;
    @SerializedName("profile")
    Profile profile;

    public Author(String email, String phone, Long profileId);
}
```

Table 37. Möbius annotated data model of Author

And a profile has a firstName, lastName, bio property, that are serialised by “firstName”, “lastName”, and “bio” accordingly

```
public class Profile {
    @SerializedName("firstName")
    String firstName;
    @SerializedName("lastName")
    String lastName;
    @SerializedName("bio")
    String bio;
    @SerializedName("consent")
    Boolean consent = false;
    @OneToOne
    @SerializedName("actor")
    Actor actor;

    public Profile(String firstName, String lastName, String bio, Boolean consent, Long actorId);
}
```

Table 38. Möbius annotated data model of Profile

The resulting JSON output is then shown below (note that each method can programmatically decide which properties of the data should be exposed in the output; this helps not to accidentally reveal critical user data e.g., passwords).

```
{
    "authors": [
        {
            "email": "john@example.com",
            "phone": "8987234",
            "profile": {
                "firstName": "John",
                "lastName": "Smith",
                "bio": "Lorem Ipsum Test Bio of John"
            }
        },
        {
            "email": "ava@example.com",
            "phone": "98761234",
            "profile": {
                "firstName": "Ava",
                "lastName": "Gibson",
                "bio": "Lorem Ipsum Test Bio of Ava"
            }
        },
        {
            "email": "thomas@example.com",
            "phone": "2986534",
            "profile": {
                "firstName": "Thomas",
                "lastName": "Anderson",
                "bio": "Lorem Ipsum Test Bio of Thomas"
            }
        }
    ]
}
```

Table 39. Möbius JSON response for getAuthorRecommendations(Long userId)

7 User feedback and integration process

The project is underpinned by a co-creation methodology, where technical developments are always followed up by the involvement of users in order to gather feedback.

In fact, after this initial “Preparatory phase” that lasts until M9, the project will undergo three further phases:

- Month 9 to month 12: Pilot Phase 1 - Concept and prototype developments. During this phase the initial concept and prototype of the prosumer intelligence toolkit and the Möbius book will be created, discussed and improved through co-creation workshops which could gather up to 10 people and follow the living lab practices. The plan is to carry out such co-creation workshops in Belgium, Spain, Italy and Germany with prosumers and publishers. Three co-creation workshops will be carried out for the

Prosumer Intelligence Toolkit. Four co-creation workshops will be organized for the Möbius book.

- Month 13 to month 18: Pilot Phase 2 - Prototype refinement and evaluation. During this phase the prototypes will be further developed and a larger number of users will be involved in each pilot country to test and refine them. The aim will be to have at least 60 people involved in evaluating the prosumer intelligence toolkit, using think-aloud methodology as well as observations and interviews, and at least 300 people in evaluating the Möbius book, gathering feedback using surveys, in-depth interviews and focus groups
- Month 19 to month 36: Pilot phase 3 - Open pilot and experimental productions. During this final phase even more users (at least 600 in total) will continue to evaluate the Prosumer Intelligence Toolkit and the Möbius book in more real-life scenarios. Feedback gathered in this way will be used for implementing further improvements. It is planned that at month 30 a final version of the software will be released; after this release and until the end of the project, the technical work will focus on the operation and support as the results are demonstrated at various events and locations.

For more information on the living lab methodology that is at the core of the project’s co-creation approach please consult deliverable D2.1 “Möbius theoretical framework: opportunities, benefits, and risks”.

We will apply agile methodologies such that the outcome of each phase is high-quality and itself forms the basis of the final solution. In this way usable tools are produced early in the software life cycle, while additional functionality can be added later. User feedback gathered is assessed regularly and translated into functional or system requirements. Thus, the initial user architecture presented in the previous section, will evolve as implement and test the various services and system as a whole. In practice, the feedback will lead to either filing of “bug” issues or feature requests issues, which are added to a backlog for each one of the services described in the Möbius architecture. The backlog of each service is assessed regularly, in order to decide which issues to add to the next development sprint.

After Phase 1, we will use continuous deployment and integration practices, essentially releasing every new good build to the users. In order to carry out improvements to the underlying services and overall system without risking crashing the running system, IN2 and EUT will set up development, staging and production servers. The ensuring workflow is as follows: once new developments are ready and work well on the dev server, they are tested on the staging server, which replicates all the conditions found on the production environment. In this way we make sure that new changes, that might work well on their own, do not bring about any negative side-effects once deployed in operational conditions. Only after this additional testing is complete will the new changes be made available to the Möbius users on the production servers.

8 Conclusion

This deliverable sets the foundations for the technical developments of the Möbius tools. It gathered the different background technologies, tools, libraries and standards that are relevant for the developments. The expected technology outcomes of the project were afterwards presented: the Prosumer Intelligence Toolkit, the Möbius Creator's Toolkit and the Möbius Player. The user requirements gathered were then summarised and analysed. Requirements that were out of scope were identified and for all the others it was the implementation implications were distilled based on three main aspects: backend, frontend and storage. Based on this, for each of the Möbius tools the functional and system requirements were specified. In total 44 requirements were identified for the backend components, 40 requirements were identified for the user interfaces and 28 requirements were identified for the storage. All these requirements were prioritised. Additionally, 19 further requirements were extracted for the Möbius container format. Architectural diagrams for each of the Möbius envisioned outcomes were presented as well as API descriptions for the data models and related methods. Finally, the approach for user feedback and integration process was presented.