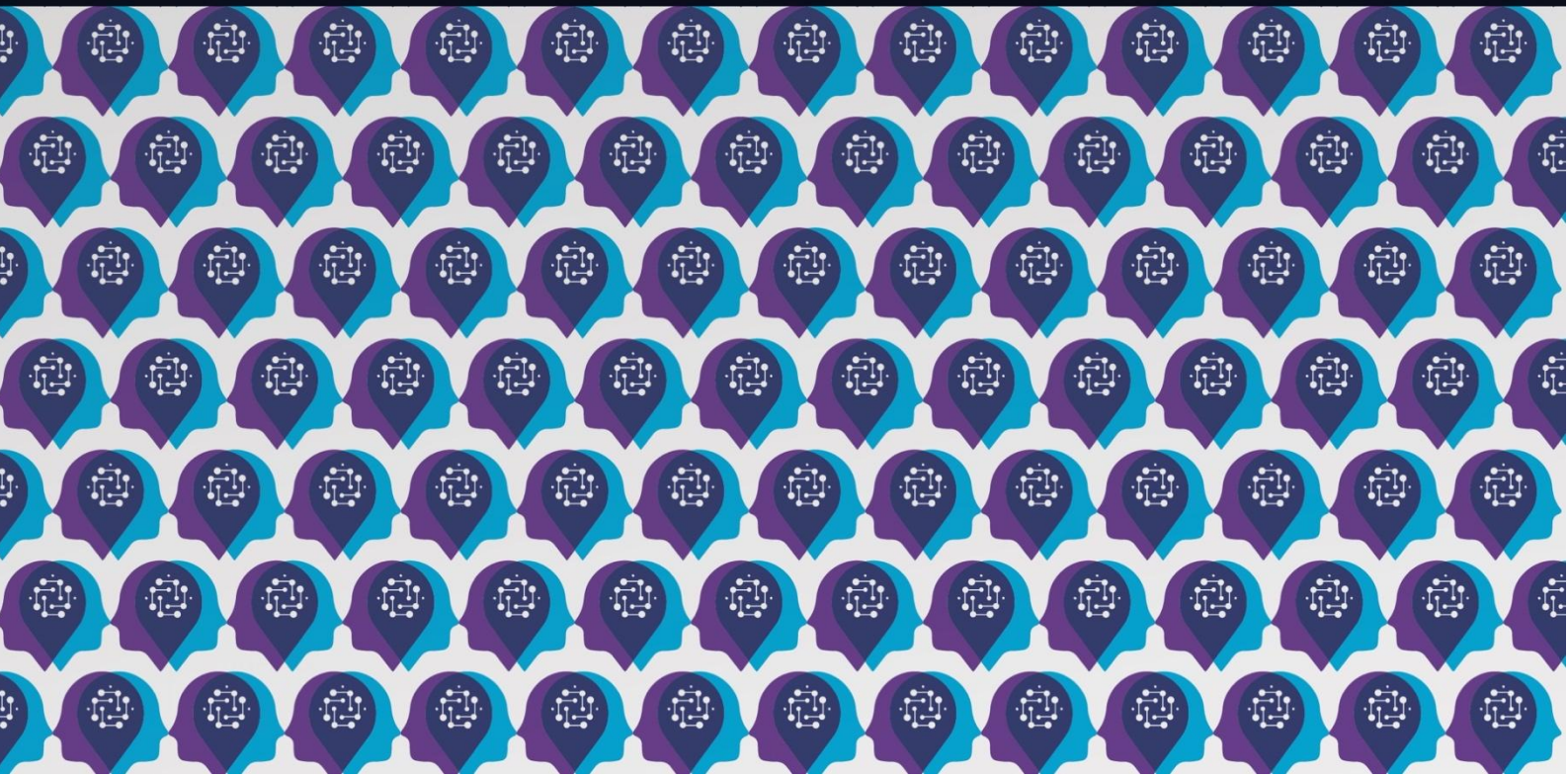
AI4Debunk

D10.1 Report on the definition of the debunking API

February 2026

Funded by
the European Union



Grant Agreement No.: 101135757
Call: HORIZON-CL4-2023-HUMAN-01-CNECT
Topic: HORIZON-CL4-2023-HUMAN-01-05
Type of action: HORIZON Innovation Actions

D10.1 REPORT ON THE DEFINITION OF THE DEBUNKING API

Project Acronym	AI4Debunk
Project Number	101135757
Project Full Title	Participative Assistive AI-powered Tools for Supporting Trustworthy Online Activity of Citizens and Debunking Disinformation
Work package	WP 10
Task	Task 10.1
Due date	28/02/2026
Submission date	23/02/2026
Deliverable lead	Hogeschool Utrecht (HU University of Applied Sciences Utrecht)
Version	1.0
Authors	Chun Fei Lung (HU), Franc van der Bent (HU)
Contributors	Despina Elisabeth Filippidou (DOTSOFT), Georgios Karanasios (DOTSOFT), Jan Kragt (IP), Kevin El Haddad (UMons), Liza Borysova (IP), Marcel Keijzer (IP), Mohammed El Amine Mokhtari (UMons), Pascaline Gaborit (Pilot4DEV), Roberto Caldelli (CNIT), Stefano Berretti (MICC/UNIFI)
Reviewers	Viktoriya Dimova (F6S), Ana Rita Alves (F6S)
Abstract	This report presents the design and development of a disinformation debunking API that can be used by software developers building user interfaces for combatting disinformation. The report outlines the API's system architecture and how it integrates with other AI4Debunk components. It also discusses how the software will be distributed to partners and potential future users.

Funded by the European Union

Keywords	disinformation, misinformation, debunking, factchecking, web engineering, application programming interface, microservices, enterprise integration, distributed systems

DOCUMENT DISSEMINATION LEVEL

Dissemination level	
X	PU - Public
	SEN - Sensitive

DOCUMENT REVISION HISTORY

Version	Date	Description of change	List of contributor(s)
0.1	06/01/2026	First complete, edited draft version for internal review by contributors who are inexplicably still at work during the winter holidays.	HU
0.2	09/02/2026	Final draft for internal review by F6S.	HU
1.0	23/02/2026	Revised version after internal revie by F6S.	HU

STATEMENT ON MAINSTREAMING GENDER

The AI4Debunk consortium is committed to including gender and intersectionality as a transversal aspect in the project's activities. In line with EU guidelines and objectives, all partners – including the authors of this deliverable – recognise the importance of advancing gender analysis and sex-disaggregated data collection in the development of scientific research. Therefore, we commit to paying particular attention to including, monitoring, and periodically evaluating the participation of different genders in all activities developed within the project, including workshops, webinars and events but also surveys, interviews and research, in general. While applying a non-binary approach to data collection and promoting the participation of all genders in the activities, the partners will periodically reflect and inform about the limitations of their approach. Through an iterative learning process, they commit to plan and implement strategies that maximise the inclusion of more and more intersectional perspectives in their activities.

Funded by the European Union

DISCLAIMER

The AI4Debunk project has received funding from the European Union's Horizon Europe Programme under the Grant Agreement No. 101135757.

Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

COPYRIGHT NOTICE

© AI4Debunk – All rights reserved
No part of this publication may be translated, reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the written permission of the publisher or provided the source is acknowledged.

How to cite this report: Lung C.F., Van Der Bent, J.F. (2026). AI4Debunk report on the definition of the debunking API. Link from website when deliverable is public.

The AI4Debunk consortium is the following:

Participant number	Participant organisation name	Short name	Country
1	LATVIJAS UNIVERSITATE	UL	LV
2	EUALIVE	EUALIVE	BE
3	PILOT4DEV	P4D	BE
4	INTERNEWS UKRAINE	IUA	UA
5	CONSIGLIO NAZIONALE DELLE RICERCHE	CNR-IRPPS	IT
6	UNIVERSITA DEGLI STUDI DI FIRENZE	MICC/UNIFI	IT
6.1	CONSORZIO NAZIONALE INTERUNIVERSITARIO PER LE TELECOMUNICAZIONI	CNIT	IT
7	BARCELONA SUPERCOMPUTING CENTER CENTRO NACIONAL DE SUPERCOMPUTACION	BSC	ES
8	DOTSOFT OLOKLIROMENES EFARMOGES DIADIKTIOY KAI VASEON DEDOMENON AE	DOTSOFT	EL
9	UNIVERSITE DE MONS	UMONS	BE
10	NATIONAL UNIVERSITY OF IRELAND GALWAY	NUIG	IE
11	STICHTING HOGESCHOOL UTRECHT	HU	NL
12	STICHTING INNOVATIVE POWER	IP	NL
13	F6S NETWORK IRELAND LIMITED	F6S	IE

Funded by the European Union

TABLE OF CONTENTS

Funded by the European Union

Funded by the European Union

LIST OF FIGURES

Funded by the European Union

LIST OF TABLES

Funded by the European Union

LIST OF LISTINGS

Funded by the European Union

ABBREVIATIONS

AI	Artificial Intelligence
AIP	API Improvement Proposal
API	Application Programming Interface
BFF	Backend for Frontend
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CIA	Confidentiality, Integrity, Availability
CI/CD	Continuous Integration and Continuous Delivery
eTLD+1	Effective Top-Level Domain + 1
EUPL	European Union Public License
GPL	GNU General Public License
HATEOAS	Hypermedia as the Engine of Application State
HTML	Hypertext Markup Language
ISO	International Organization for Standardization
JWT	JSON Web Token
JSON	JavaScript Object Notation
LLM	Large Language Model
ML	Machine Learning
OSI	Open Software Initiative
REST	Representation State Transfer
RFC	Request for Comments
S3	Simple Storage Service
SaaS	Software as a Service
SHA256	Secure Hash Algorithm (with 256-bit digests)
UML	Unified Modeling Language
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
WP	Work Package
YAML	YAML Ain’t Markup Language

Funded by the European Union

EXECUTIVE SUMMARY

AI4Debunk is a European initiative that aims to develop tools that help citizens fight disinformation. The initiative includes the development of AI-powered modules that support this goal, such as detecting potential signs of deepfakes, identifying incorrect captioning, and finding sources that support or refute a claim. AI4Debunk also provides an educational VR app and three human-centred software applications that allow users to check and report content for disinformation: a browser extension, a smartphone app, and a collaborative platform. This report describes the design of the debunking API, a piece of software that acts as an intermediary between these three latter user interfaces and the AI-powered modules, ensuring that access to the modules is simple, stable, and secure.

The design of the debunking API is informed by an iterative design process that takes its requirements from the needs of AI4Debunk's three human-centred software applications, years of industry experience, and well-established technical standards. The debunking API conceptually consists of two parts. A core framework handles integration with current and future user interfaces (the API), security, data persistence, and the creation of long-running asynchronous debunking tasks. This framework is complemented by a set of module-specific adapters that integrate the debunking API with the AI-powered modules developed alongside the API as part of AI4Debunk.

The software will be distributed in two ways: as a freely accessible online service throughout the lifetime of the AI4Debunk project and as free software licensed under the European Union Public License (EUPL-1.2). Users wishing to install the software themselves will be able to use the software via a publicly available container image or build their own copy from source.

Funded by the European Union

1 INTRODUCTION

AI4Debunk is a European initiative that aims to develop human-centred, multimodal, collaborative AI tools to fight disinformation and protect democratic values. Its human-centred solutions include four distinct interfaces: a browser extension, a smartphone app, a collaborative platform, and an educational virtual reality (VR) app that explores the use of a prebunking approach.

The first three interfaces enable ordinary European citizens to directly assess the accuracy of online content using a combination of debunking modules that incorporate state-of-the-art AI/ML models and a human-in-the-loop mechanism designed to enhance data quality and foster greater trust in the debunking process.

Many of these AI/ML models require such significant computing resources that they cannot run even on high-end consumer devices, let alone mobile devices with limited hardware specifications such as laptops and smartphones. Moreover, developing software that runs correctly on a wide range of different devices can be an insurmountable challenge.

For these reasons, we do not run AI/ML models locally on end-users' devices. Instead, we use a client-server architecture in which clients – software such as browser extensions and smartphone apps – can request services from a central server (Van Vliet, 2008, p. 259) that provides access to debunking modules. This architecture shifts computation and data management from clients to the server, allowing clients to remain lightweight and simplifying installation, maintenance and management of the debunking modules.

In a client-server architecture, the client and server exchange messages over a network. The server runs an AI4Debunk software service that exposes a web application programming interface (API) to clients. This API is essentially a remote user interface for software applications that can use the API to perform various tasks (Lauret, 2019, p. 2), such as authentication and content submission.

For simplicity reasons, we henceforth refer to the software service as the debunking API, even though the actual API itself is not its main distinguishing feature. The actual raison d'être for the debunking API is to intermediate between clients and the debunking modules. In software parlance, the debunking API decouples clients and debunking modules, allowing them to evolve independently from each other. It also ensures that access to the debunking modules is simple, stable, and secure.

In this report we provide a high-level description of the debunking API and outline the architectural considerations that guide its design and implementation.

Funded by the European Union

1.1 OVERVIEW

The remainder of this report is structured as follows. Section 2 provides an overview of academic and grey literature on the design and implementation of APIs. Based on best practices and recommendations from this literature, Section 3 outlines the overall design goals of the debunking API and breaks these down into individual requirements. Section 4 presents a high-level design of the debunking API, which aims to meet these requirements and explains how it facilitates communication between clients and the API's backing services, i.e. the AI-powered debunking modules responsible for tasks related to debunking. Section 5 then describes how we plan to make the system easily deployable across a diverse set of environments. Finally, Section 6 reflects on the work completed so far as part of WP 10, discusses the limitations of our current approach, and identifies the design work that remains for WP 11.

Funded by the European Union

2 BACKGROUND

Modern web APIs that integrate multiple software components – especially when these are developed independently – typically employ a microservice architecture and are designed with cloud-native deployment methods in mind. In this section, we briefly explain these concepts and how each logically follows from another.

2.1 WEB APIS

An application programming interface (API) allows software components, often developed by different teams or organisations, to communicate with each other. The most common type of API takes the form of reusable software libraries, which provide functions such as sending an email or calculating a standard deviation. These libraries are installed locally, either as part of an application or alongside it.

Web APIs serve a similar purpose but differ in how they are accessed and controlled. Unlike libraries, which users install and manage themselves, web APIs are accessed remotely over a network. This means that users have limited control over updates: when the provider modifies a web API, changes are generally imposed on all clients. This approach benefits the API provider, as it allows them to maintain complete control over functionality and protect sensitive information such as passwords or personal data. It can also benefit users, because computational requirements – such as processing large datasets or performing complex calculations – are handled by the provider rather than the client, reducing resource demands on the user's system (Geewax, 2021, pp. 4–5).

Most modern web APIs follow the Representational State Transfer (REST) architectural style introduced by Fielding (2000). REST is based on the concept of resources – entities such as products or documents – that can be read and manipulated using standard HTTP verbs. APIs that adhere to REST principles are referred to as RESTful, meaning they implement the constraints defined by the REST architecture, such as statelessness and uniform interfaces. A key principle of REST is that web APIs should be navigable in a way that resembles how humans browse websites.

To standardise RESTful API design, several guidelines have been proposed, including JSON:API[1] and Google's API Improvement Proposals[2] (Ahmad et al., 2024). However, even after 13 and 9 years respectively since their introduction, neither has achieved widespread adoption. This limited uptake is often attributed to factors such as complexity, competing standards, and lack of awareness. Best practices have also been described in professional literature, notably in *The Design of Web APIs* by Lauret (2019) and *API Design Patterns* by Geewax (2021), which offer alternative less prescriptive recommendations.

[1] https://jsonapi.org/
[2] https://google.aip.dev/

Funded by the European Union

In practice, many APIs are not fully REST-compliant or even RESTful; instead, they follow conventions that only superficially resemble REST (Koschel et al., 2019). A significant portion of the web development community combines various good practices, such as publishing OpenAPI[3] specifications. These specifications describe available endpoints, request formats, and other details, and are widely used because they enable automated generation of documentation and integration code, improving interoperability and developer experience.

2.2 MICROSERVICE ARCHITECTURES

Traditionally, web APIs were developed as monoliths – single software applications whose modules cannot be executed independently (Dragoni et al., 2017). From a deployment perspective, monoliths are difficult to use in distributed environments, where software runs on multiple machines that together form a single system to achieve scalability or resiliency, rather than on a single machine (Abgaz et al., 2023).

From a development perspective, monoliths present several challenges. Their large size and complexity make them difficult to understand, maintain, and evolve. They often suffer from "dependency hell", where adding or updating libraries can lead to inconsistent systems that fail to compile or run, or worse, misbehave. Any modification requires restarting the application, which for large monoliths can take considerable time and render the system unavailable to clients. Deployment is also problematic because the single server hosting the monolith must accommodate the requirements of all components, which may conflict or differ significantly, resulting in a one-size-fits-all configuration that is either costly or forces some components to run sub-optimally. Finally, monoliths impose technology lock-in, as all components must use the same programming language and framework.

Microservice architectures were designed to address these issues by treating applications not as a single software package but as a distributed system composed of small, independent components called microservices. These microservices communicate over a network using messages. Each microservice is self-contained, making it easy to develop, test, and understand. Because they are independent, microservices can be deployed separately, scaled individually, and hosted in environments that best suit their requirements.

However, microservices also introduce new challenges, many which stem from the characteristic described earlier in Section 2.1: web APIs impose upgrades on clients consuming them (Lercher et al., 2024). New API versions may introduce breaking changes that clients cannot handle, causing crashes or incorrect behaviour. Mitigating these issues often requires extensive communication and coordination

[3] https://www.openapis.org/

Funded by the European Union

between microservice developers and the teams responsible for clients, whether user interfaces or other microservices.

To reduce the complexity of integrating numerous independently evolving microservices, backend systems are often exposed through an API gateway. The gateway aggregates, transforms, and routes requests to underlying services, allowing clients to interact with a single entry point rather than multiple APIs (Newman, 2015). This approach significantly reduces the communication overhead and simplifies integration.

When an API must serve many different clients, another challenge arises: responses may either be overly generic or highly specific, creating tight coupling between clients and the API. A common solution is to implement backends for frontends (BFFs), where each client-facing team develops its own backend tailored to its specific needs (Newman, 2015). This pattern improves flexibility for client teams but can also introduce additional complexity at the system level, as multiple backends must be developed and maintained.

2.3 CLOUD-NATIVE COMPUTING

Containerisation (also referred to as Dockerisation, named after Docker, the product that first popularised containerisation technologies) is a lightweight virtualisation technology that encapsulates software and its dependencies in an isolated environment (Muzumdar et al., 2024). This approach enables the creation of portable software that can run across diverse environments, whether on developers' workstations or in production environments serving real users. Because containerisation simplifies development, testing, and deployment of software in isolation, it is particularly well-suited for microservice architectures (Mikkonen et al., 2022).

Beyond its role in microservices, containerisation is highly valuable for ensuring reproducibility[4], as it allows software artefacts to be packaged consistently – without which experimental results obtained using the software are difficult to reproduce. At the same time, containerisation is a cornerstone of cloud-native software engineering, which focuses on producing software explicitly designed to run in private, hybrid, and public clouds such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). Mitchell (2023) defines cloud-native applications as well-architected systems that are containerised and dynamically managed. In this context, "well-architected" refers to systems that adhere to established software engineering practices while leveraging capabilities offered by cloud environments; "containerised" describes the packaging and deployment of software using technologies such as Docker, Docker Compose, Kubernetes, and serverless platforms; and "dynamically managed" involves making

[4] Technically, containerisation by itself does not ensure full bitwise reproducibility yet, but from a practical standpoint it is close enough.

effective use of cloud resources, including durable block storage, horizontal auto-scaling, and security services.

Concrete guidance for designing and implementing cloud-native applications can be found in the Twelve-Factor App methodology, a set of principles originally intended to teach developers how to build applications that can be automated and setup declaratively, maintain a clean contract with the underlying operating system, and scale dynamically (Hoffman, 2016). Since its original publication, additional rules and guidelines have emerged on topics such as building "API first", concurrency, security, and telemetry.

2.4 INTEGRATED OPERATIONS

Traditionally, the development, delivery, and operation of software were carried out in separated silos. DevOps is a set of practices designed to integrate software development and operational deployment, fostering collaboration and shared responsibility among all technical stakeholders for delivering high-quality software (Colomo-Palacios et al., 2018).

A core principle of DevOps is automation, which is commonly achieved through Continuous Integration and Continuous Delivery (CI/CD). CI/CD involves continuously merging small changes into the main codebase, followed by automated testing and deployment (Elazhary et al., 2021). These processes are typically supported by CI tools, such as GitHub Actions, Travis CI, and Jenkins (Golzadeh et al., 2022).

For cloud-native applications that incorporate AI/ML models, MLOps, extends DevOps principles to machine learning workflows. MLOps is a set of practices, tools, and technologies that streamline the integration of ML into DevOps (Zarour et al., 2025). Key practices include the automation of CI/CD pipelines for ML models, orchestrating of tasks and data flows, implementing robust versioning, and continuously monitoring model performance to ensure reliability and accuracy.

2.5 SECURITY

Security is a critical quality requirement for any software deployed in real-world environments. Unfortunately, it is often treated as an afterthought, addressed via static analysis or penetration testing – after code has already been written. To ensure robust protection, security must be integrated into the software development process itself. Software security involves applying sound engineering principles and considering security early in the software lifecycle (McGraw, 2006). This includes planning for authentication and authorisation mechanisms, and ensuring resilience against malicious actors who may attempt to manipulate, disable, or steal information from the system.

For systems that handle sensitive data, information security is particularly important. Goodrich and Tamassia (2014) define information security using the CIA triad: confidentiality, integrity, and availability. Confidentiality refers to preventing unauthorised disclosure of information, such as protecting user-

provided data. Integrity ensures that information is not altered in an unauthorised manner; for example, ordinary users should not be able to modify other users' data. Availability guarantees that authorised users can access and modify information in a timely manner, even under adverse conditions, e.g. the system should remain resilient against denial-of-service attacks.

Funded by
the European Union

3 REQUIREMENTS

Unlike the other four interfaces in WP 10, the debunking API is primarily aimed at software developers and system integrators in need of an easy way to programmatically access the debunking modules over a remote network.

A practical consequence of this is that the debunking API exists mainly to address technical and organisational challenges. Its requirements are therefore not derived directly from end-user needs but instead emerge from the constraints imposed by its clients – described in more detail by Lung and Van Der Bent (2026), DOTSOFT (2026), and Kragt and Keijzer (2026) – and by backing services, as outlined by El Haddad (2025a) and El Haddad et al. (2025). Other important sources include recommendations from social sciences and humanities (SSH) work packages, D5.3 input and output requirements (Berretti & Caldelli, 2025b), the operational environments in which the debunking API is expected to function, and relevant technical standards for web applications. At the same time, the debunking API also imposes indirect requirements on the debunking modules it invokes, ensuring compliance with technical standards and client expectations, as well as on user interfaces, so that all functionality provided by backing services is accessible to end users.

To structure our requirements, we first formulate a set of four overarching design goals. We then decompose these into individual functional and quality requirements.

3.1 DESIGN GOALS

To ensure that the API is easy to use, develop, and operate, its system architecture must at the very least meet the following design goals:

1. Provide a simple, stable, and secure way for developers of the three AI4Debunk user interfaces that are being concurrently developed alongside the API and any other future applications to access debunking models and other services related to content analysis.
2. Decouple clients from backing services such that each stakeholder can freely evolve their deliverables without the need to coordinate and synchronise change activities with everyone else.
3. Be resilient to changes in the external environment, including but not limited to unavailability of compute or cloud services, the introduction of new or improved AI models, and sudden spikes in legitimate and adversarial traffic.

These three design goals emerge organically from the goals of AI4Debunk as stated on the project website – namely, to develop AI-powered solutions that help European citizens decipher fact from fiction (AI4Debunk, 2024) – and the contemporary industry best practices for software engineering in a web development context that were previously described in section 2.

Funded by the European Union

3.2 USER STORIES

In this section, we describe the requirements for the debunking API using user stories, which follow the template in Table 1. User stories are widely adopted in agile software development to incorporate user and business requirements into the development process (Lucassen et al., 2016). They are considered more effective than UML use cases for creating a shared understanding of a problem domain (Dalpiaz & Sturm, 2020) and are easier to apply than goal-oriented requirements engineering methods that are primarily used in academic contexts, such as i* and KAOS (Mavin et al., 2017). The user stories presented here were primarily gathered through a combination of technical meetings and consultations with developers of client applications (WP10.2–10.5) and debunking modules (WP6–9). In this process, we adopted the perspectives of both contractor (for client applications) and customer (of debunking modules), while simultaneously acting as a neutral system integrator. We also analysed relevant technical documentation within and beyond the AI4Debunk project to ensure sufficient coverage of critical requirements.

This approach ensures that the context of use, as defined in ISO/IEC 25002 (2024) – including users, their goals, the user environment, and the system context – is properly considered. Each user story is assigned a unique identifier for traceability throughout the software development lifecycle, labelled with its provenance if it is derived from deliverable 5.3 (Berretti & Caldelli, 2025b), and tagged as either a functional or quality requirement. Functional requirements describe the capabilities needed by an actor with a particular role to solve a problem or achieve an objective (Van Vliet, 2008, p. 202), i.e. what actions the system should perform or support. Quality requirements[5] are based on the ISO/IEC 25010 (2023) standard, which defines product quality in terms of nine characteristics: functional suitability, performance efficiency, compatibility, interaction capability, reliability, security, maintainability, flexibility, and safety. These requirements primarily apply to the debunking API itself, but because it depends on modules developed by other partners within the AI4Debunk consortium, some requirements also apply transitively to those modules – or at least to the API's integration with them.

Note that within the context of the debunking API, roles are often fulfilled by software applications or the organisations developing them. To avoid formulating user stories with rationales such as "so <organisation> can build their app", we express them from the perspective of the end user. This makes the context of use clearer. We use the generic term "user" to refer to end users of any human-centred AI4Debunk interface. In cases where requirements originate from specific user interfaces, we indicate the source interface(s) in parentheses, even though any interface can technically use any feature.

[5] Quality requirements are often referred to as non-functional requirements. However, we find that term unnecessarily confusing because it describes what they are not rather than what they are.

Funded by the European Union

TABLE 1: COMPONENTS OF A USER STORY TEMPLATE

Purpose	Template	Example
Role	As a(n) …	**As an** Inner Party employee at the Ministry of Truth
Goal	I want …	**I want** to monitor the thoughts of Outer Party members
Benefit	so that …	**so that** I can proactively suppress dissent against the Party.

We prioritise user stories using the first three MoSCoW priority levels, which Van Vliet (2008, p. 59) defines as follows: **"must haves"** are required for a working system, **"should haves"** are important but not absolutely needed for a usable system, and **"could haves"** are only implemented if time permits.

3.2.1 CONTENT ANALYSIS

The primary purpose of the disinformation debunking API is to support client application developers in building features that help end users analyse and report content suspected of containing disinformation. To achieve this, the API must be able to receive content, process it, and return results in a form that aligns with the needs of those client applications.

MUST FUNC D5.3

R1 As a user
I want to **know if content in local text or multimedia files contains signs of disinformation**
so that I can avoid sharing or relying on misleading information.

MUST FUNC D5.3

R2 As a user (of the smartphone app or collaborative platform)
I want to **know if online content located at a remote URL contains signs of disinformation**
so that I can make informed decisions before trusting or sharing it.

MUST FUNC D5.3

R3 As a user (of the browser extension)
I want to **know if online content embedded within a web page contains signs of disinformation**
so that I can evaluate the credibility of what I read while browsing.

MUST FUNC D5.3

R4 As a user
I want to **receive a disinformation score for submitted content that is easy to understand**
so that I can quickly see how likely it is that the content contains signs of disinformation.

MUST FUNC D5.3

R5 As a user
I want to **receive an explanation of how the disinformation score was calculated**
so that I can better understand why and whether the content contains disinformation.

Funded by the European Union

MUST FUNC D5.3

R6 As a user
I want to **know if an audio, image, or video file is potentially a deepfake**
so that I can verify its authenticity before using or sharing it.

MUST FUNC D5.3

R7 As a user
I want to **know if a caption accurately describes an image**
so that I can detect misleading or false visual narratives.

MUST FUNC D5.3

R8 As a user
I want to **know if a textual claim is supported by other sources**
so that I can confirm its reliability and avoid spreading disinformation.

MUST FUNC

R9 As a user
I want to **flag content containing disinformation to AI4Debunk**
so that it can be shared with others.

MUST FUNC

R10 As a user
I want to **view the report once it has been validated by AI or a committee**
so that I can confirm whether the flagged content contains disinformation and why.

MUST FUNC

R11 As a user
I want to **explicitly make my reports publicly available**
so that others can benefit from its findings and avoid disinformation.

SHOULD FUNC

R12 As a user
I want to **keep my reports private**
so that findings about the submitted content are not shared without my consent.

MUST QUAL

R13 As a system administrator
I want **users' reports to be publicly available by default unless specified otherwise**
so that they are searchable for other users.

SHOULD FUNC

R14 As a user
I want to **delete a validated report that contains information I want to keep hidden**
so that I can maintain control over my contributions and remove incorrect information.

Funded by the European Union

SHOULD FUNC

R15 As a user
I want to **report mistakes in a validated report**
so that the errors can be corrected by a human expert.

MUST FUNC

R16 As a user
I want to **find validated reports by keyword, topic, date, verdict, and/or language**
so that I can quickly locate relevant information for my research or decision-making.

SHOULD QUAL

R17 As a user
I want **search queries to take no more than one second under typical circumstances**
so that I can efficiently access information without unnecessary delays.

MUST QUAL

R18 As a system administrator
I want **users' debunking requests to be processed sequentially**
so that the system is not overwhelmed when many requests are made simultaneously.

SHOULD QUAL

R19 As a system administrator
I want **the system to be able to process debunking requests in parallel**
so that the system can better cope with high traffic volumes.

MUST FUNC

R20 As a software developer of the browser extension
I want to **control the layout of the extension's user interface via the API**
so that I can immediately push updates to the extension's user interface.

MUST FUNC

R21 As a user (of the browser extension)
I want to **know the reliability of a news source if it is known**
so that I can estimate the reliability of content presented by it.

MUST FUNC D5.3

R22 As a user (of the browser extension)
I want to **know if a URL is already known to contain disinformation**
so that I can avoid opening it altogether.

COULD FUNC

R23 As a user
I want to **receive a notification as soon as an analysis is complete**
so that I know when I can view the results.

Funded by
the European Union

MUST QUAL

R24 As a user
I want **the analysis to be completed in a reasonable amount of time**
so that I do not have to wait excessively before making decisions based on the result.

MUST FUNC

R25 As a client
I want **the system to return all raw data that is available**
so that I can choose what information I want to present to the user.

MUST QUAL D5.3

R26 As a client
I want **data returned with precise, semantic values rather than presentation-oriented labels**
so that I can format and display it as needed.

3.2.2 FILE MANAGEMENT

End users of client applications often want to submit multimedia and other forms of rich content for analysis. These files can be quite large, which makes them harder to handle and increases the likelihood that they contain sensitive information requiring careful protection. As a result, file handling and processing introduce an additional set of requirements that the API must address.

MUST FUNC D5.3

R27 As a user
I want to **upload audio, image, video, and text files to the API**
so that I can have them checked for signs of disinformation.

MUST FUNC

R28 As a user
I want to **check if a file has already been uploaded to the API**
so that I do not waste bandwidth uploading it twice.

MUST QUAL

R29 As a system administrator
I want to **enforce a limit on the maximum size of uploaded files**
so that the server does not run out of disk space.

MUST FUNC

R30 As a system administrator
I want **uploaded files to be deduplicated**
so that the amount of storage space required to run the software can be minimised.

Funded by the European Union

MUST FUNC

R31 As a system administrator
I want **uploaded files to be automatically removed when they are no longer needed**
so that the amount of storage space required to run the software can be minimised.

MUST FUNC

R32 As a user
I want **my uploaded files to be automatically removed when they are no longer needed**
so that my data is not stored permanently and my privacy is protected.

SHOULD FUNC

R33 As a user
I want to **actively delete a file that I have uploaded**
so that it does not stay on a server longer than necessary.

3.2.3 PRIVACY AND SECURITY

Security and privacy are essential for any software that aims to protect its users' wellbeing. This subsection outlines user stories that ensure that the debunking API processes data securely, respects end users' privacy, and adheres to best practices in responsible data handling, while at the same time providing enough flexibility for technical users such as developers who may not require the same degree of protection.

MUST QUAL

R34 As a software developer
I want **to use the API without any form of authentication**
so that I can easily interact with it in local and secure environments.

SHOULD QUAL

R35 As a system administrator
I want **the API to rate-limit requests**
so that a single user cannot easily overload the system.

MUST QUAL

R36 As a system administrator
I want **debunking requests from the public to be authenticated**
so that I can control who is (still) allowed to use the system's resources.

MUST QUAL

R37 As a system integrator
I want **users to authenticate via an provide-agnostic open stateless protocol**
so that users can log in via my own existing portals and applications.

Funded by the European Union

MUST FUNC

R38 As a system integrator
I want **authentication to work for both human and non-human users**
so that the system can interoperate with other information systems.

COULD FUNC

R39 As a system administrator
I want to **generate a complete export of a user's personal data with a single command**
so that I can comply with GDPR Article 15 (right of access by the data subject).

COULD FUNC

R40 As a system administrator
I want to **permanently delete a user's personal data using with a single command**
so that I can comply with GDPR Article 17 (right to erasure).

MUST QUAL

R41 As a privacy officer
I want **the system to return as little personal information as possible**
so that users' privacy is protected.

MUST QUAL

R42 As a security officer
I want **the system to hide information that is not necessary to implement features**
so that I can minimise the risk of information about the system and its users being leaked.

MUST QUAL

R43 As a security officer
I want **the system to meet the OWASP ASVS[6] level 1 requirements**
so that most common attacks against it can be prevented.

SHOULD QUAL

R44 As a security officer
I want **the system to meet the OWASP ASVS level 2 requirements**
so that the system is sufficiently resilient against common and less common attacks.

[6] The OWASP Application Security Verification Standard as defined on https://owasp.org/.

Funded by the European Union

3.2.4 DEVELOPMENT AND OPERATIONS

The debunking API, as a software product primarily aimed at technical users, must not only be easy to use but also provide straightforward ways for developers and system administrators to deploy it, operate it, and diagnose technical issues.

SHOULD FUNC

R45 As a software developer
I want to **view human-readable, up-to-date OpenAPI documentation**
so that I know what endpoints I can use when building my software.

SHOULD FUNC

R46 As a software developer
I want to **download an up-to-date OpenAPI specification in JSON format**
so that I can automatically generate code that interacts with the API.

MUST QUAL

R47 As a system integrator
I want **the system to be easily deployable in traditional hosting environments**
so that I can integrate it with existing infrastructure without major changes.

MUST QUAL

R48 As a system integrator
I want **the system to be easily deployable in cloud-native environments**
so that I can leverage scalability and flexibility offered by modern cloud platforms.

MUST FUNC

R49 As a system integrator
I want **the system's AI modules to be usable via cloud inference providers**
so that I can avoid large upfront capital expenditures.

MUST FUNC

R50 As a system integrator
I want **the system's AI modules to be usable in an in-house private cloud**
so that I can avoid uploading sensitive information to a third-party server.

MUST FUNC

R51 As a system administrator
I want **the system to log errors with detailed a stack trace**
so that I can diagnose and resolve technical issues efficiently.

Funded by the European Union

MUST FUNC

R52 As a system administrator
I want **the system to log domain events**
so that I can inspect the recent state of the system.

MUST FUNC

R53 As a system integrator
I want **the system to distribute domain events via a message broker**
so that other applications can respond to events without being tightly coupled to the system.

COULD FUNC

R54 As a system administrator
I want **access to real-time metrics**
so that I can continuously monitor the performance of the system.

SHOULD QUAL

R55 As a third-party software developer
I want **the source code to follow established design and coding practices**
so that I can easily understand it and contribute changes and fixes to the software.

MUST QUAL

R56 As a third-party software developer
I want **the system to be modular**
so that I can easily add, upgrade, and override components.

Funded by
the European Union

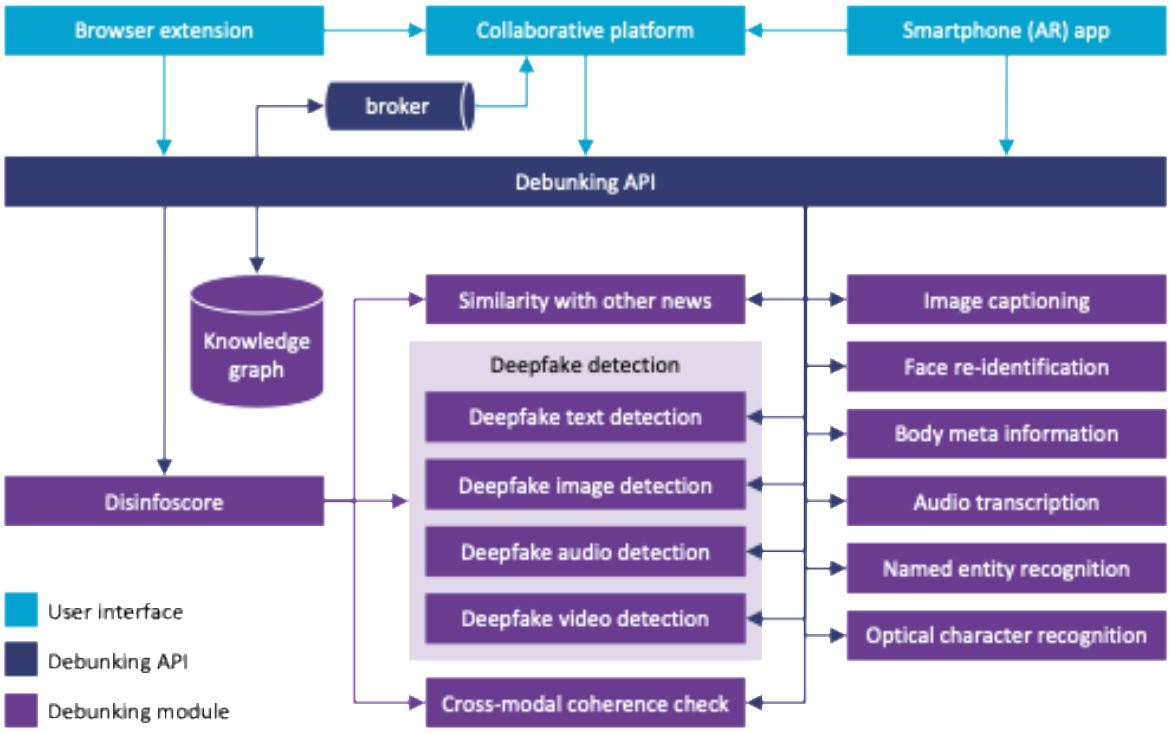


FIGURE 1: OVERALL SYSTEM ARCHITECTURE WITH ALL COMPONENTS (THAT WE CURRENTLY KNOW OF)

4 SYSTEM DESCRIPTION

The debunking API acts as an API gateway that provides simple, stable, and secure access to the backing services developed as part of WP6–9. It is designed with two primary use cases in mind: first, as a service operated by the AI4Debunk consortium for the duration of the project; and second, as a standalone product that can be installed and managed by third parties after the project concludes.

This section describes the overall system architecture of the debunking API, its main components, and how these components interact with clients and backing services. While the implementation may evolve over time due to new insights or evolving requirements, the architecture outlined here is expected to remain applicable to the final version.

4.1 SYSTEM ARCHITECTURE

The debunking API plays a central role in the overall system architecture of the AI4Debunk platform, as illustrated in Figure 1.

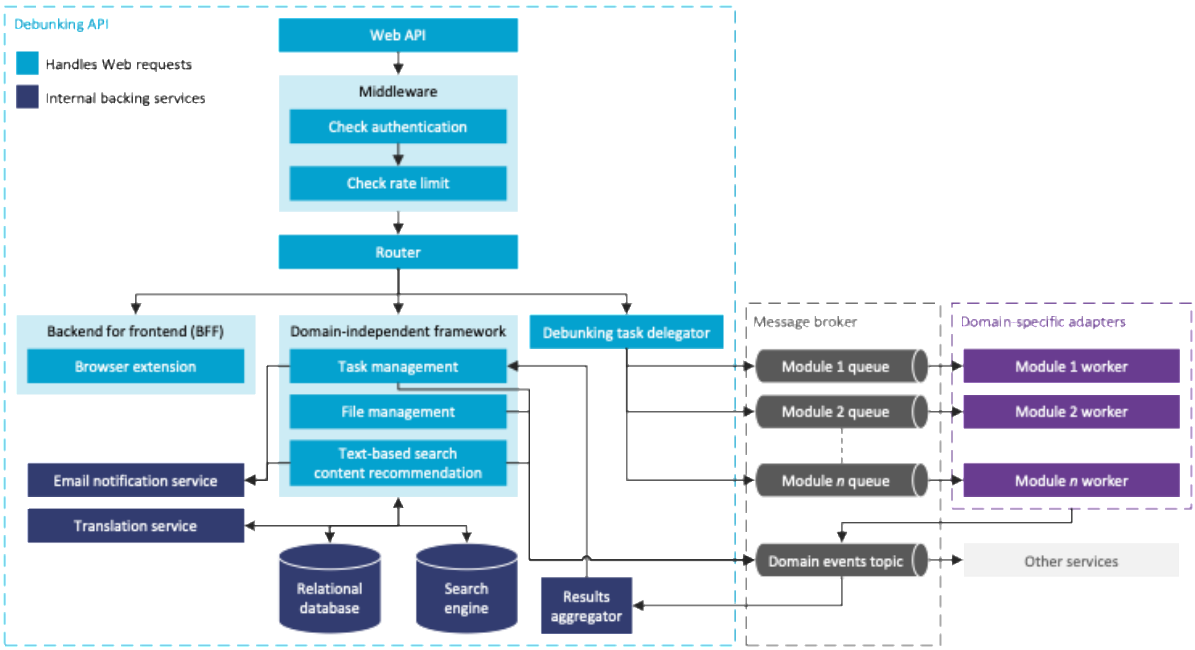


FIGURE 2: INTERNAL DEBUNKING API ARCHITECTURE

At the core of the AI4Debunk system is a knowledge base (El Haddad et al., 2025), and set of AI-powered debunking modules capable of detecting potential signs of disinformation in submitted content (Berretti & Caldelli, 2025a) and extracting relevant metadata (El Haddad, 2025a). The project also includes several clients in the form of human-centred user interfaces that interact with these modules. Connecting each user interface directly to every AI-powered module would require approximately $M \times N = 3 \times 15 = 45$ integrations, where M is the number of clients and N the number of backing services. This would be costly to develop and maintain. Instead, access to these modules is mediated through the debunking API, reducing the number[7] of required integrations to $M + N = 3 + 15 = 18$. This approach not only simplifies integration but also ensures compliance with quality requirements such as suitability, reliability, and security. Additionally, domain events are distributed via a message broker, enabling the API to actively push information to clients without creating tight coupling. As a result, clients do not need to be aware of each other's existence, and the same applies to the AI-powered modules.

Figure 2 depicts the internal architecture of the debunking API, which consists of two main parts. The first is a domain-independent framework that provides core functionality for executing and monitoring long-running tasks with text and multimedia-based payloads. Conceptually, this framework includes middleware that acts as a "border control", processing incoming requests and verifying whether they should be allowed to proceed, as well as a set of endpoints backed by a database and other services for functionality such as file uploads and task management. The second part comprises domain-specific

[7] This number does not include the integrations required for the disinformation score module (El Haddad, 2025b).

Funded by the European Union

adapters that integrate with external debunking modules. These adapters can run alongside the API or the module. By default, the API includes the modules developed within the AI4Debunk project, but it can be dynamically extended to support third-party modules in the future.

The remainder of this section provides more detailed explanations of specific components: the core framework, integration with AI/ML modules developed in WP6–9, a web content extraction module for processing user-provided URLs, and a client-specific API that functions as an integrated backend for frontend (BFF).

4.2 DOMAIN-INDEPENDENT CORE

The domain-independent core of the debunking API provides essential functionality that enables simple, stable, and secure access to backing services.

4.2.1 AUTHENTICATION

The debunking API can operate in either authenticated or unauthenticated mode (R34, R36). Unauthenticated mode is intended for deployments on a single machine, within internal networks, or behind a web application firewall (WAF). When the API is exposed to the public internet and used by clients such as browser extensions and smartphone applications, authentication and usage-based rate limiting can be enabled via environment variables. This enables operators to configure the API in a range of modes, from free and unrestricted access, to configurations where unauthenticated requests are permitted but subject to strict rate limits, to configurations in which the rate limit for unauthenticated requests is set to 0, thereby requiring all requests to be authenticated.

Initially, the debunking API supports one method of authentication: JSON Web Tokens (JWT). Authentication can be enabled or disabled using environment variables. JWT is an open standard that allows the exchange of digitally signed claims (R37) between two systems (Jones et al., 2015). In the current design, the collaborative platform generates a digitally signed JWT containing key user information, such as the user identifier. Clients send this JWT as a Bearer token in the HTTP `Authorization` header. The API verifies that the token's signature to confirm authenticity, enabling user authentication without requiring detailed knowledge of the user or the system that issued the JWT (R38). This authentication flow is illustrated in the sequence diagram in Figure 3.

4.2.2 RATE LIMITING

To prevent malicious users from overloading the system, the debunking API includes rate-limiting functionality (R35). By default, rate limiting is disabled, which is appropriate when the debunking API runs as a private instance on a local machine or within a corporate network. When enabled, the limit applies

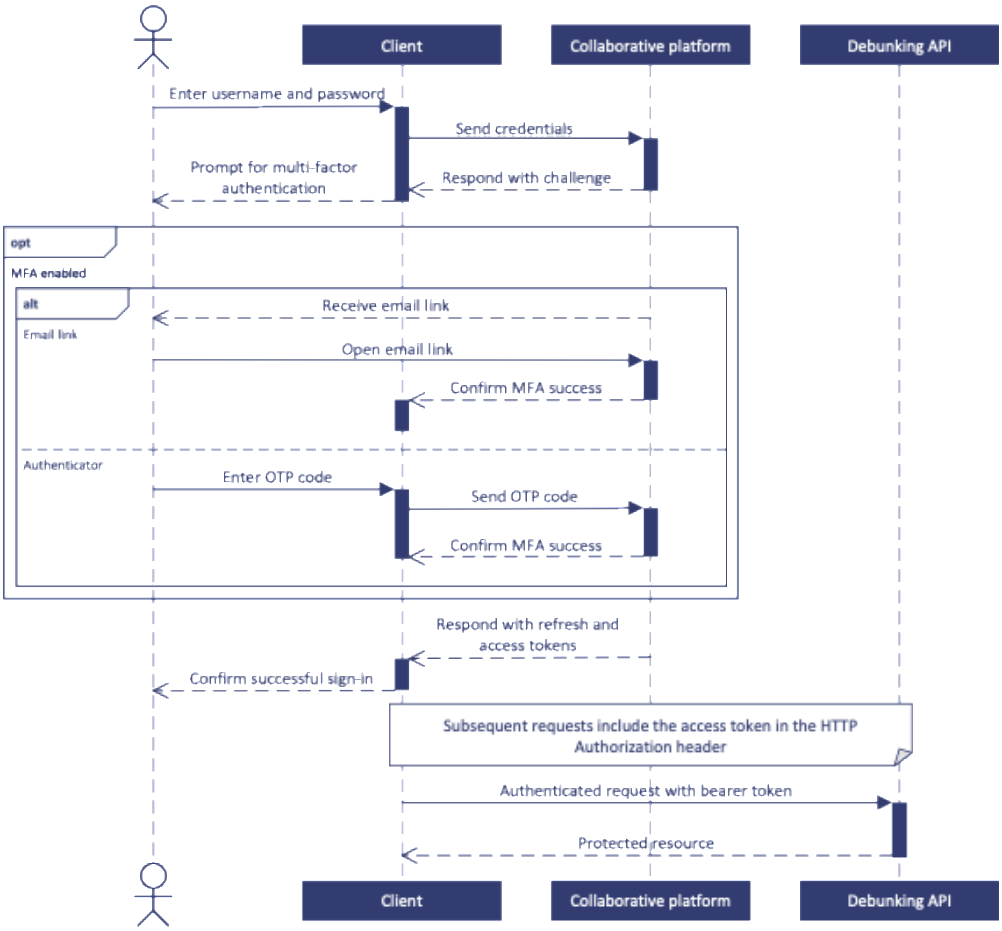


FIGURE 3: SEQUENCE DIAGRAM OF AUTHENTICATION FLOW

globally to all unauthenticated requests. If authentication is active, rate limiting is applied per user, with the highest configured value taking precedence.

4.2.3 ASYNCHRONOUS TASK PROCESSING

Some debunking modules may require significant processing time to complete tasks. This impacts the response times of the debunking API and, consequently, the perceived responsiveness of user interfaces that submit these requests on behalf of users. The system may also become overloaded if it must handle many computationally tasks simultaneously.

There are three common approaches for handling long-running HTTP requests:

1. **Blocking requests**: Clients wait until the request completes. In a blocking implementation, the client cannot perform other actions during this time. If the connection fails unexpectedly, the request is lost and must be retried. This approach typically results in a poor user experience.

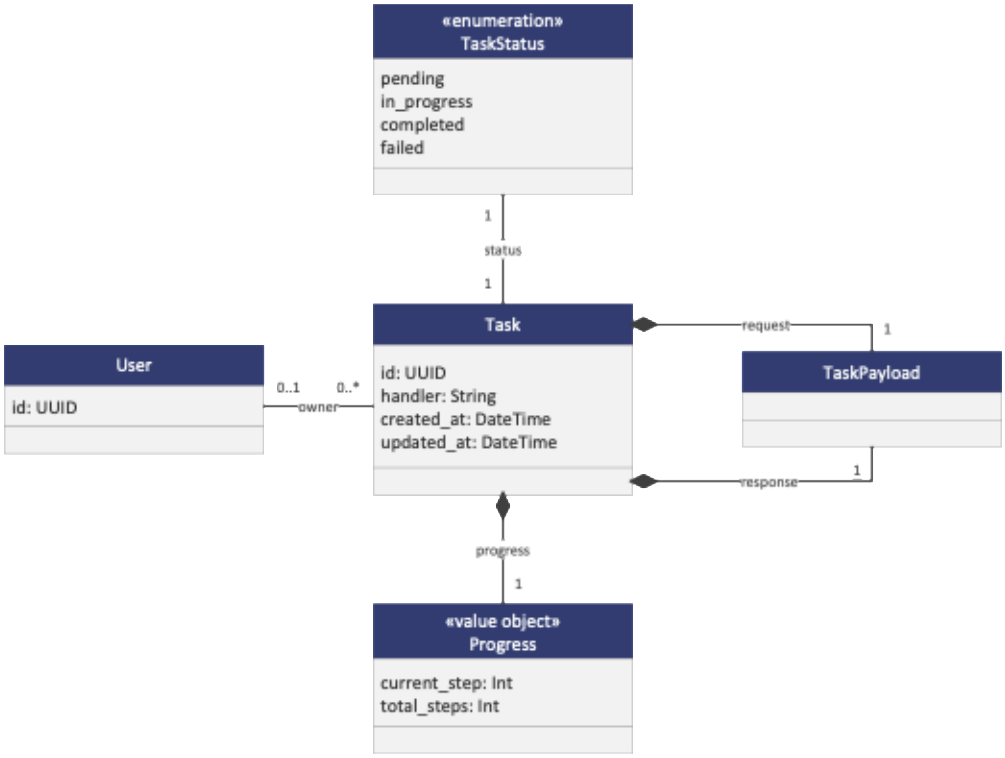


FIGURE 4: TASKS DOMAIN MODEL

2. **Callbacks**: The server notifies the client when the task is complete using mechanisms such as callback URLs, WebSockets, or push notifications. This is the most energy-efficient method for handling long-running operations over HTTP (Rani et al., 2024). However, each mechanism has drawbacks: callback URLs are generally unsupported by browser extensions and mobile apps, WebSockets require a persistent internet connection; and push notifications introduce dependencies on third-party services such as Google or Apple.

3. **Polling**: As described in Google's AIP-151 for long-running operations[8], the API returns a handle (similar to a receipt) after the initial request and queues a job to work on the operation in the background (R18). The client then polls a dedicated endpoint using this handle until the result is available. This approach adds slight complexity to both client and server but offers flexibility, resilience to connectivity issues, and independence from third-party services.

The third approach best aligns with our design goals, as it is simple to implement, robust against network failures, and avoids external dependencies.

[8] https://google.aip.dev/151

Funded by the European Union

Figure 4 illustrates the domain model for tasks in the debunking API. The Task entity represents operations initiated by clients on behalf of users and stores metadata such as the task type, payload (parameters), task status, and response (if available). A task may be associated with a user.

Clients create tasks via domain-specific endpoints, for example, to check a video file for deepfake content. Upon creation, the API returns a UUIDv4 identifier. We opted for UUIDv4 over alternatives such as UUIDv7 or ULID because it makes it harder for malicious actors to infer information about the number or timing of tasks. The identifier acts as a receipt or tracking code that clients use to query a generic tasks endpoint. Depending on the task status, the endpoint may indicate that it the task is either queued or in progress (in which case the client continues polling) and otherwise completed or failed (in which case the response includes the result). Tasks cannot be deleted but can be cancelled while queued and soft-deleted once they are in progress or completed.

4.2.4 FILE UPLOADS

The debunking API supports various types of file uploads (R27), including text, audio and video (Berretti & Caldelli, 2025b). Some of these files may be very large, requiring significant time and bandwidth to upload, and considerable storage space, even if only temporarily. Uploading files can therefore be inconvenient to both the API and end users.

To mitigate these issues, the API is designed to minimise resource usage while respecting user privacy. Figure 5 illustrates the domain model for file uploads, which consists of two entities: Files and Uploads.

A File entity represents a physical file and stores metadata that can be inferred directly from the file, such as its checksum, size in bytes, MIME type, and storage location. In line with the Twelve-Factor App principles, file storage is treated as a backing service (Hoffman, 2016). Storage may be a physical disk or a remote object storage service such as Amazon S3[9] or an S3-compatible alternative. Storage configuration

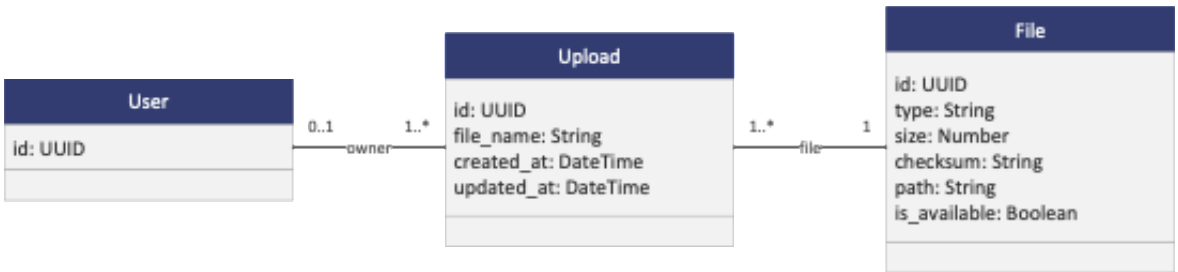


FIGURE 5: FILE UPLOADS DOMAIN MODEL

[9] https://aws.amazon.com/s3/

Funded by the European Union

is applied application-wide and does not need to be stored within the File entity. To ensure files are not retained longer than necessary, a housekeeping process periodically removes physical files that have not been accessed within a defined time frame or when storage capacity is low (R31, R32). This process deletes only the physical file, not the File entity. An attribute within the File entity indicates whether the physical file is currently available. If a user wishes to use a File that has been removed, they must re-upload it.

An Upload entity contains metadata specific to a user's instance of the file, such as the original filename and creation date. Upload entities are created when a client uploads a file for the first time and updated whenever the same client interacts with that file.

Distinguishing between Files and Uploads provides an important benefit: when multiple clients upload the same file, only one copy needs to be stored (R30), while each uploader sees a data transfer object representing a singular file with their own familiar name and timestamps (R41).

There are a few privacy-related considerations. First, clients cannot directly delete a physical file, as it may still be required by other users. Instead, they can request deletion of their Upload entity (R33), which removes the file from their view but retains the physical file until no other user needs it. The system does not disclose whether other users are still using the file to protect privacy. Similarly, different users who upload the same file must always upload it themselves; otherwise, subsequent user would learn that the file was previously uploaded, potentially revealing sensitive information about other users (R41).

4.2.5 TELEMETRY

A running system should not be an opaque box: developers and operators must be able to inspect its current and historical state to understand behaviour, diagnose runtime failures, and audit user actions (Li et al., 2020). Other components may also need to subscribe to domain events that capture significant occurrences within the application's business domain (Vernon, 2013, p. 285). Additionally, operators may want to monitor key metrics such as CPU usage and response times to assess system performance.

The debunking API therefore incorporates features that expose telemetry for operational monitoring (R52) and business logic execution (R53).

The most basic type of telemetry is logging. In accordance with the Twelve-Factor App principles, logs should ideally be treated as an event stream that is written to `stdout` and `stderr`, enabling automatic capture and analysis (Hoffman, 2016). For operators running the API in a traditional hosting environment, support for rotated log files is also provided.

Logs typically feed into observability platforms for debugging or monitoring anomalous behaviour. Domain events are like logs in that they record significant events, but they are also distributed to other system components that subscribe to them and take appropriate action. Given the adoption of a microservice

Funded by the European Union

architecture, domain events will be distributed via a message broker such as RabbitMQ[10] for enterprise application integration (Hohpe & Woolf, 2004). Operators can configure domain event distribution and destinations via environment variables.

To support metric monitoring, the debunking API optionally exposes a `/metrics` endpoint that can be scraped periodically by cloud-native tools such as Prometheus[11] to retrieve performance data (Turnbull, 2018). The endpoint can be enabled by setting the appropriate environment variable.

4.3 DEBUNKING MODULES

The debunking API it intended to be deployed alongside a set of debunking modules developed in WP6–9. These modules are not part of the debunking API itself, as the API merely mediates access to them. However, some knowledge of the modules is required to integrate them properly. Furthermore, because the modules are developed independently, transformations may be necessary between the inputs and outputs of the API and those of the modules.

The Adapter pattern, first proposed by Gamma et al. (1995, p. 139), is an object-oriented design pattern that enables classes to "work together that could not otherwise because of incompatible interfaces". While the original definition assumes that adapters encapsulate locally available objects (Shalloway & Trott, 2002), the debunking API follows a cloud-native paradigm, meaning its adapters may encapsulate resources exposed as backing services over a network.

Adapters implement the integration between the debunking API and individual debunking modules. They interact with both but are not strictly part of either. We propose a modular architecture that allows modules to be dynamically added to or removed from the API without requiring source code changes to either the module or the API (R56).

Each adapter functions as a worker process that handles messages conforming to an AsyncAPI[12] specification. It accepts all or a subset of the inputs defined by Berretti and Caldelli (2025b) and returns the result in JSON format, with semantics documented using an AsyncAPI specification. The adapter runs as a containerised application that embeds, invokes, or otherwise interacts with the module, ensuring proper translation of requests and responses. Typically, the module is deployed in a sidecar container[13] alongside the adapter container. If this is not feasible or desirable – and the module's software license permits – it may also run within the adapter container itself.

[10] https://www.rabbitmq.com/
[11] https://prometheus.io/
[12] https://www.asyncapi.com/
[13] https://kubernetes.io/docs/concepts/workloads/pods/sidecar-containers/

Funded by the European Union

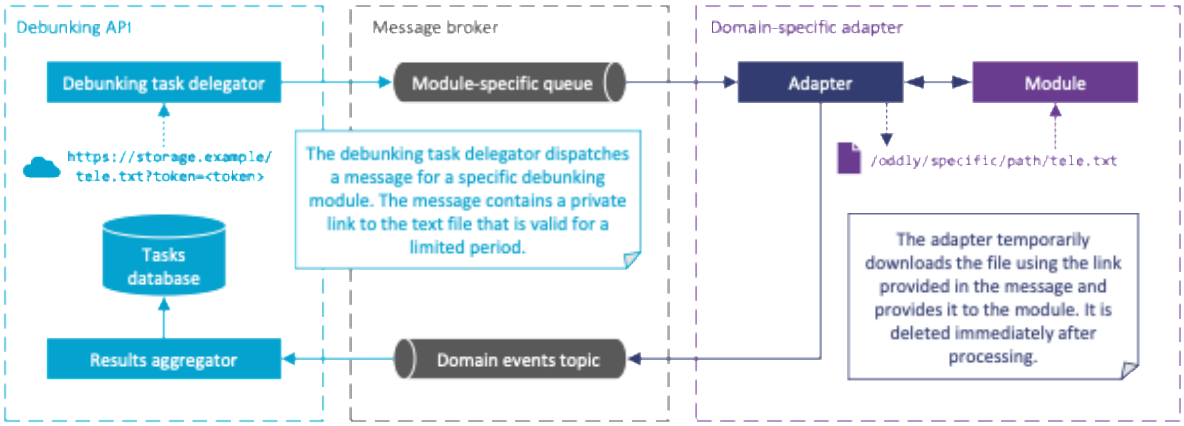


FIGURE 6: MODULE-SPECIFIC ADAPTERS

To illustrate how this mechanism works, consider a debunking module that expects a text file located in a local directory `/oddly/specific/path` and outputs a numeric value. In contrast, the debunking API may provide files via remote cloud storage. To bridge this gap, the adapter ensures the file is available at `/oddly/specific/path` by making it available locally. It then maps the API request to the module's expected format, waits for the module to produce a result or error, or to time out, and forwards the outcome to the debunking API in the appropriate format by submitting it to the message broker. Figure 6 visualises this integration process.

4.4 WEB CONTENT EXTRACTION

All three human-centred interfaces developed alongside the debunking API support the submission of web pages. For the browser extension, this will always be the page currently open in the user's browser. For the smartphone app and the collaborative platform, the user typically submits a URL. The AI4Debunk system must be able to process these web pages and infer what the user wants to check (R2, R3). This introduces two main challenges related to web scraping: retrieving the web page and extracting its main content.

The first challenge does not apply to the browser extension, as it already has access to the HTML content of the submitted page. For the other two interfaces, however, the API receives only a URL and must retrieve the page content itself. Normally, this can be done using tools such as `curl`. However, as Wahed et al. (2024) note, many websites employ techniques such as IP blocking, login walls, and CAPTCHAs to prevent automated scraping. Even when technically possible, ethical and legal constraints may apply – for example when a website's `robots.txt` file or terms of service prohibit scraping. At present, it is unclear to what extent the debunking API will encounter such issues.

The second challenge concerns extracting the relevant HTML content for the user's debunking request, especially when no additional context is provided beyond the URL. HTML pages are structured documents

Funded by the European Union

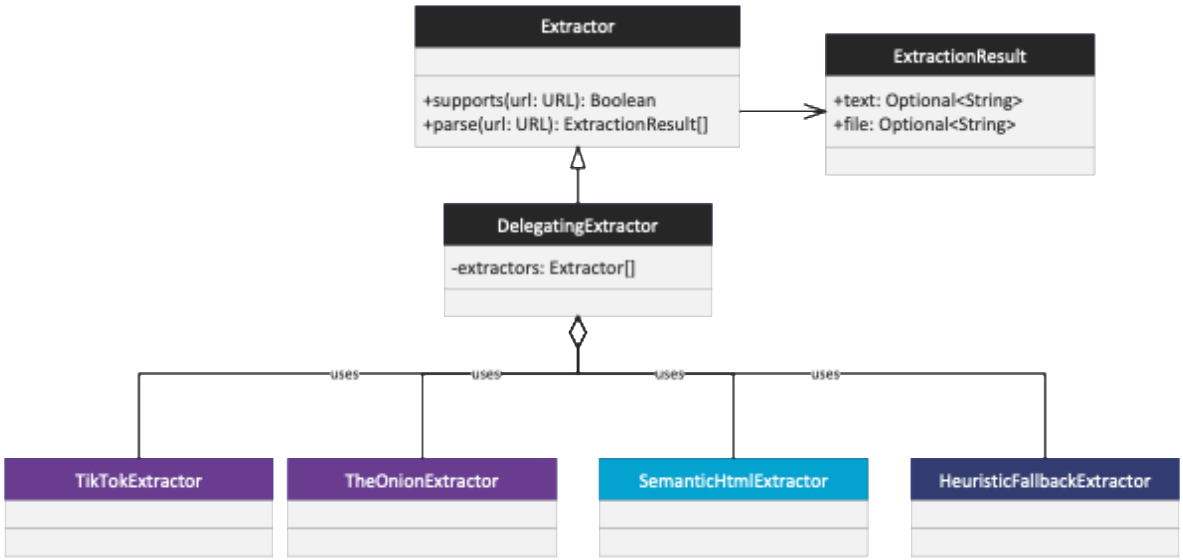


FIGURE 7: CLASS DIAGRAM SHOWING THE MODULAR CONTENT EXTRACTION SYSTEM

that typically follow conventions optimised for search engines, enabling the use of heuristics and statistical models for main content extraction. Bevendorff et al. (2023) provide an overview of fourteen popular libraries for extracting main content from web pages, while Dallabetta et al. (2024) propose a rule-based approach tailored to specific websites, trading versatility for accuracy. LLMs may also be used for content extraction, but their accuracy and cost-effectiveness remain uncertain. Furthermore, many LLMs are non-deterministic – identical inputs may produce different outputs – posing challenges for transparency and reproducibility (Sallou et al., 2024) and for LLMs used via cloud providers availability is also an issue. Adding complexity, user-provided URLs may reference pages where the main content is loaded dynamically using client-side JavaScript or is embedded in a video stream (e.g. `https://www.youtube.com/watch?v=PYlIbalC7P0&t=20s`). The web content extraction module should be able to recognise such cases and attempt to extract the relevant segment of the video.

We propose a modular system for content extraction, illustrated in Figure 7. The system uses a list of extractors, with site-specific, semantic, and heuristic extractors. Each extractor encapsulates a third-party open-source component capable of processing content. Extractors are executed sequentially until content is successfully extracted or all fail. This principle is also applied in the browser extension, albeit with fewer and simpler extractors (Lung & Van Der Bent, 2026).

From the debunking API's perspective, web content extraction is treated similarly to a debunking module (R56): it is developed as a separate project and installed alongside the API. This design provides flexibility, allowing the component to be used independently and integrated into third-party software. It also facilitates contributes from external developers and enables easy future replacement with improved versions if necessary.

Funded by the European Union

4.5 CLIENT-SPECIFIC API FOR THE BROWSER EXTENSION

The debunking API serves three human-centred interfaces that share substantial functionality but also require interface-specific features. The browser extension, in particular, depends on several endpoints that are unique to it: the ability to dynamically construct its user interface based on capabilities provided by the debunking API, and the ability to automatically retrieve the trustworthiness of a source based on the domain, URL or content of the current web page (Lung & Van Der Bent, 2026). To support these requirements, the debunking API provides a set of BFF-style endpoints specifically intended for use by the browser extension.

4.5.1 DYNAMIC USER INTERFACE

Software distributed through app stores must undergo a review process before publication. This process can introduce delays, preventing timely rollout of updates to users. To address this issue, the browser extension's user interface is not hardcoded. Instead, the debunking API exposes an entry endpoint that defines the structure of the user interface, the functionality it supports, and the static content to display (R20). Using pre-built user interface components, the browser extension dynamically constructs its interface based on this entry point, allowing operators to update the UI without relying on third-party approval.

This mechanism is conceptually similar to the Hypermedia as the Engine of Application State (HATEOAS) constraint for REST architectures proposed by Fielding (2000). HATEOAS is based on the principle that a REST API should provide hyperlinks to other endpoints that clients can invoke based on their current application state – similar to how human users navigate a website by following links from the home page.

4.5.2 SOURCE TRUSTWORTHINESS

A key feature of the browser extension is its ability to display the trustworthiness of news sources (R21) based on the domain name directly below the effective top-level domain (eTLD+1), e.g. `rainews.it` and `bbc.co.uk`.

Trustworthiness indicators – such as factuality, credibility, and political bias – are available from sources like Media Bias/Fact Check[14] (MBFC) and NewsGuard[15] (Burdisso et al., 2024) and are common used in research projects addressing disinformation (Jahanbakhsh & Karger, 2024). Comparable datasets have also been published on platforms such as Kaggle. The debunking API provides an endpoint that retrieves trustworthiness information for a web page based on the eTLD+1 of the user's current browser tab.

[14] https://mediabiasfactcheck.com/
[15] https://www.newsguardtech.com/

Funded by the European Union

5 DISTRIBUTION

The debunking API is distributed using two primary models. The first is a Software as a Service (SaaS), in which a hosted instance of the API is provided by the AI4Debunk consortium for direct access by the browser extension and smartphone app. The second model involves distributing the source code and compiled software artefacts as free software, allowing technical users to download, install, configure, and maintain the API themselves.

5.1 SOFTWARE AS A SERVICE

A hosted production instance of the debunking API will be made available under a SaaS model to clients at `api.ai4debunk.chuniversiteit.eu`, along with all required backing services, for the entire duration of the project and for a period to be specified after its conclusion. This instance is primarily intended for use by the three human-centred debunking interfaces – the browser extension, smartphone app, and collaborative platform – to enable evaluation of their effectiveness. However, it can also be accessed by third-party developers wishing to explore the API. The service is provided free of charge but is subject to resource availability.

5.2 FREE SOFTWARE

We intend to release the debunking API software to the public under version 1.2 of the European Union Public License (EUPL-1.2). The EUPL is an OSI-approved free and open-source copyleft license designed to be legally robust within the European Union and compatible with other major open-source software licenses such as the GPL (Schmitz, 2013).

The license applies only the debunking API, its adapters, and any installers that we provide. It does not cover the backing services on which it depends, as these are governed by their own respective software licenses. This separation is achieved by excluding backing service source code and artefacts from the API distribution and deferring their inclusion until installation or deployment. This approach is common practice and provides technical users with greater flexibility. At the time of writing, not all backing services have confirmed licenses, and some rely on software that prohibits commercial use. Consequently, we are not yet able to provide an installer that fully meets legal requirements.

The software can be run in several ways. Although it is technically possible to run the debunking API natively without containers using Python virtual environments, we do not officially support this method because such installations are difficult to troubleshoot due to subtle environment differences. Instead, we recommend running the API and its adapters in containers, which will be cross-published to Docker Hub and the GitHub Container Registry (GHCR). While users can manually configure containers, deployment is typically managed using container orchestration tools such as Docker Compose and Kubernetes.

Funded by the European Union

5.2.1 DEPLOYMENT USING DOCKER COMPOSE

Docker Compose[16] is the recommended deployment method for running the debunking API on a single machine for development, demonstration, or evaluation purposes. Docker Compose simplifies the definition and deployment of multi-container applications and their associated network and storage configurations (Docker, 2025). Its ease of use and integration with Docker have made it one of the most widely adopted container orchestration tools (Mikkonen et al., 2022).

We provide a main `compose.yaml` file that launches the debunking API and its required backing services with a single command. Users can customise the configuration by editing the YAML files directly or by providing a `compose.override.yaml` file.

5.2.2 CLOUD-NATIVE DEPLOYMENT ON KUBERNETES

For production environments spanning multiple servers, Kubernetes[17] is the preferred deployment platform. Kubernetes is an industry-standard container orchestration system that automates the deployment, scaling, and management of containerised applications in large-scale environments (Junior et al., 2022). Its primary goal is to improve availability and maintainability. For example, Kubernetes can deploy multiple instances of the same container across servers to distribute load and can automatically recover from failures by reallocating workloads when a node becomes unavailable.

Applications are typically configured for Kubernetes using YAML files that declaratively define container deployment and lifecycle management. A single default configuration is unlikely to meet all operator requirements. Helm, a package manager for Kubernetes, addresses this by enabling package maintainers to package configurations into Helm charts (R48), which allow easy customisation (Zerouali et al., 2023). Users can install these charts on their own Kubernetes platforms using Helm.

[16] https://docs.docker.com/compose/
[17] https://kubernetes.io/

Funded by the European Union

6 DISCUSSION

Most of the initial design work on the AI4Debunk disinformation debunking API has now concluded. This report has outlined its requirements, overall system architecture, and intended distribution methods. These design decisions emerged over the past year through discussions with consortium partners, exploratory prototyping, and reviews of scientific and professional literature.

Official implementation of the debunking API is scheduled to begin in March 2026 (M27). In practice, the groundwork for the final version has already been laid through three experimental prototypes developed to evaluate architectural concepts and integrations with modules provided by AI4Debunk partners. These experiments have revealed two key bottlenecks that must be addressed in the definitive version to be developed under WP11.

Technology maturity is the first bottleneck. The latest Python-based prototype uses a mix of mature and emerging open-source components, notably FastAPI[18] for the client-facing API, SQLModel[19] for working with the relational database, and Celery[20] for asynchronous task processing. FastAPI and Celery were selected for their popularity in the Python ecosystem, while SQLModel is recommended by FastAPI. However, these components lack the maturity required for a system aiming for TRL 7. For example, FastAPI and SQLModel do not provide many features available in more comprehensive frameworks such as Django[21], forcing developers to rely on third-party modules that may not integrate seamlessly. Furthermore, both libraries are maintained by a single individual, posing a long-term sustainability risk if maintenance ceases.

The second bottleneck concerns integration with modules developed in WP6–9. Integration with third-party modules is inherently challenging and becomes even more complex when those modules are under active development and subject to interface changes. To mitigate this, we have provided a practical guide for consortium partners developing backing services for the API (see Appendix A). So far, the first partner to deliver integratable software was DOTSOFT with a GraphQL API for the knowledge graph. UMONS currently has the most integrations, while MICC was the first to deliver a container that image that fully meets our guidelines. These guidelines were formulated before finalising the complete set of requirements described in Section 3, which include performance and stability considerations, so some minor revisions may still be necessary.

[18] https://fastapi.tiangolo.com/
[19] https://sqlmodel.tiangolo.com/
[20] https://docs.celeryq.dev/en/stable/
[21] https://www.djangoproject.com/

Funded by the European Union

6.1 LIMITATIONS

The current design of the debunking API has several known limitations.

First, the API is expected to reach a TRL of 6–7 by the end of the AI4Debunk project in 2027, corresponding to "System/subsystem model or prototype demonstration in a relevant environment" or "System prototype demonstration in an operational environment" (Olechowski et al., 2020). However, the TRL of a system is determined by its weakest component. Since the API integrates multiple novel components and includes two modules with a minimum target TRL of 5 (the multimodal knowledge graph and multimodal fake news detection modules), the maximum achievable TRL for the API technically might not exceed 5. Moreover, the TRL scale focusses on component maturity rather than integration quality, which is equally critical (Sauser et al., 2010). A well-known example is the Mars Climate Orbiter failure in 1999, caused by immature integration between otherwise mature components (Héder, 2017). We do not currently anticipate any issues, as many of the relevant risks have already been mitigated, as outlined in the AI4Debunk risk plan. In the coming years, we will continue working closely with WP6–9 partners to ensure that the system as a whole can achieve its intended TRL 6–7 target, including providing support with productionisation and integration where necessary. Additionally, we have not yet considered other possible maturity assessments, such as Societal Readiness Levels, which are essential for evaluating the API's potential societal impact (Bruno et al., 2020). These concerns are partially mitigated through collaboration with social science and humanities (SSH) partners in WP12–14.

A second limitation relates to estimating source reliability. Current datasets primarily cover established news outlets and well-known sources such as satirical websites. They exclude smaller regional sources and niche platforms that may either be reliable or spread disinformation. Alternative approaches could address this gap. For example, Burdisso et al. (2024) propose reinforcement learning to estimate source reliability based on citation patterns. Another suggestion, provided by a journalist at Dutch news organisation NOS, is to focus on narratives, claims and information within the content itself, since content can be republished and otherwise reputable sources may still spread disinformation, while disreputable sources may occasionally publish accurate information (personal communication, 11 September 2025). This idea aligns with findings from social sciences and humanities and technical solutions developed within AI4Debunk and has been explored in prior projects Yousuf et al. (2021).

6.2 FUTURE WORK

While the architectural concepts outlined in this report have largely been validated, several unresolved issues remain and will be addressed under WP11.

The most critical task is to refactor the current debunking prototype to adopt the Django framework and incorporate well-integrated components from its ecosystem. This migration is expected to enhance maintainability, stability, and long-term sustainability. Django's maturity and cohesive design possibly also

Funded by the European Union

make it more suitable for LLM-assisted development, which is becoming increasingly prevalent. As this refactor primarily concerns internal architecture, it will not affect client-facing functionality or existing backing services.

Another ongoing challenge is the integration with clients and backing services, particularly ensuring that the system continues to meet its design goals and can be deployed by media organisations, political entities, and other stakeholders in production environments. Achieving this will require continuous optimisation to reduce system requirements – for example through model compression, architectural improvements, or introducing early-exit mechanisms in the processing pipeline to avoid unnecessary computation on invalid inputs (Shi et al., 2025).

Looking beyond the AI4Debunk project, strategies must be developed to ensure the API's long-term relevance. Potential directions include implementing priority-based access control to guarantee fair resource allocation, refining authentication and authorisation mechanisms, fostering adoption among technical and journalistic communities, and introducing new features to meet the needs of future users.

Funded by
the European Union

7 CONCLUSION

The AI4Debunk disinformation debunking API is designed to provide simple, stable, and secure access for the project's three human-centred interfaces. This report first presented an overview of contemporary practices in the design, development, and operation of web APIs. It then outlined the requirements derived from industry best practices and the functional constraints imposed by the API's primary client interfaces. In addition, the report described the system architecture and its main components, along with the principal distribution methods during and after the project's completion. Finally, it identifies several critical challenges that must be addressed in the forthcoming implementation phase (WP11).

Funded by the European Union

REFERENCES

Abgaz, Y., McCarren, A., Elger, P., Solan, D., Lapuz, N., Bivol, M., Jackson, G., Yilmaz, M., Buckley, J., & Clarke, P. (2023). Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review. *IEEE Transactions on Software Engineering*, *49*(8), 4213–4242. https://doi.org/10.1109/TSE.2023.3287297

Ahmad, M., Geewax, J., Macvean, A., Karger, D., & Ma, K.-L. (2024). API Governance at Scale. *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, 430–440.

AI4Debunk. (2024, March 4). *Home—AI4Debunk*. AI4Debunk. https://ai4debunk.eu/

Berretti, S., & Caldelli, R. (2025a). *Initial report on the modules developed* (D8.1; AI4Debunk).

Berretti, S., & Caldelli, R. (2025b). *Report on requirements* (D5.3; AI4Debunk).

Bevendorff, J., Gupta, S., Kiesel, J., & Stein, B. (2023). An Empirical Comparison of Web Content Extraction Algorithms. *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2594–2603. https://doi.org/10.1145/3539618.3591920

Bruno, I., Lobo, G., Covino, B. V., Donarelli, A., Marchetti, V., Panni, A. S., & Molinari, F. (2020). Technology readiness revisited: A proposal for extending the scope of impact assessment of European public services. *Proceedings of the 13th International Conference on Theory and Practice of Electronic Governance*, 369–380. https://doi.org/10.1145/3428502.3428552

Burdisso, S., Sanchez-cortes, D., Villatoro-tello, E., & Motlicek, P. (2024). Reliability Estimation of News Media Sources: Birds of a Feather Flock Together. In K. Duh, H. Gomez, & S. Bethard (Eds), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)* (pp. 6893–6911). Association for Computational Linguistics. https://aclanthology.org/2024.naacl-long.383

Colomo-Palacios, R., Fernandes, E., Soto-Acosta, P., & Larrucea, X. (2018). A case analysis of enabling continuous software deployment through knowledge management. *International Journal of Information Management*, *40*, 186–189.

Dallabetta, M., Dobberstein, C., Breiding, A., & Akbik, A. (2024). Fundus: A Simple-to-Use News Scraper Optimized for High Quality Extractions. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, 305–314. https://doi.org/10.18653/v1/2024.acl-demos.29

Dalpiaz, F., & Sturm, A. (2020). Conceptualizing Requirements Using User Stories and Use Cases: A Controlled Experiment. In N. Madhavji, L. Pasquale, A. Ferrari, & S. Gnesi (Eds), *Requirements Engineering: Foundation for Software Quality* (Vol. 12045, pp. 221–238). Springer International Publishing. https://doi.org/10.1007/978-3-030-44429-7_16

Docker. (2025). *Docker Compose*. Docker Documentation. https://docs.docker.com/compose/

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. *Present and Ulterior Software Engineering*, 195–216.

Funded by the European Union

El Haddad, K. (2025a). *First report on the development of multimodal adaptable AI modules* (D8.2; AI4Debunk).

El Haddad, K. (2025b). *Initial calculation of a score representing the amount of disinformation in the data* (D8.4; AI4Debunk).

El Haddad, K., Gotev, G., Angelova, K., Shcherba, D., Caldelli, R., Berretti, S., Kragt, J., Keijzer, M., Van Der Bent, F., Nasir, J., D'Andrea, A., & D'Ulizia, A. (2025). *First report on the process of continuous graph adaptation* (D6.4; AI4Debunk).

Elazhary, O., Werner, C., Li, Z. S., Lowlind, D., Ernst, N. A., & Storey, M.-A. (2021). Uncovering the benefits and challenges of continuous integration practices. *IEEE Transactions on Software Engineering*, *48*(7), 2570–2583.

Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures* [PhD Thesis].

Filippidou, D. E., Karanasios, G., Katsaridis, S., Katsakioris, D., Nikas, M., Simeonidou, A., Nikopoulos, G., & Maragkos, C. (2026). *Report on the definition of the smartphone app* (D10.3; AI4Debunk). DOTSOFT.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Pearson.

Geewax, J. J. (2021). *API design patterns*. Simon and Schuster.

Golzadeh, M., Decan, A., & Mens, T. (2022). On the rise and fall of CI services in GitHub. *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 662–672.

Goodrich, M. T., & Tamassia, R. (2014). *Introduction to Computer Security*. Pearson.

Héder, M. (2017). From NASA to EU: the evolution of the TRL scale in Public Sector Innovation. *The Innovation Journal*, *22*(2), 1–23.

Hoffman, K. (2016). *Beyond the Twelve-factor App: Exploring the DNA of Highly Scalable, Resilient Cloud Applications*. O'Reilly Media.

Hohpe, G., & Woolf, B. (2004). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional.

ISO/IEC. (2023). *ISO/IEC 25010:2023 Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—Product quality model* (ISO/IEC 25010:2023). ISO/IEC.

ISO/IEC. (2024). *ISO/IEC 25002:2024 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Quality model overview and usage* (ISO/IEC 25002:2024). ISO/IEC.

Jahanbakhsh, F., & Karger, D. R. (2024). A Browser Extension for in-place Signaling and Assessment of Misinformation. *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 1–21. https://doi.org/10.1145/3613904.3642473

Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT)*.

Junior, P. S., Miorandi, D., & Pierre, G. (2022). Good shepherds care for their cattle: Seamless pod migration in geo-distributed kubernetes. *2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC)*, 26–33.

Koschel, A., Blankschyn, M., Schulze, K., Schoner, D., Astrova, I., & Astrov, I. (2019). RESTfulness of APIs in the Wild. *2019 IEEE World Congress on Services (SERVICES)*, 382–383. https://doi.org/10.1109/SERVICES.2019.00114

Kragt, J., & Keijzer, M. (2026). *Report on the definition of the collaborative platform* (D10.4; AI4Debunk). Innovative Power.

Lauret, A. (2019). *The design of web APIs*. Simon and Schuster.

Lercher, A., Glock, J., Macho, C., & Pinzger, M. (2024). Microservice API Evolution in Practice: A Study on Strategies and Challenges. *Journal of Systems and Software*, *215*, 112110. https://doi.org/10.1016/j.jss.2024.112110

Li, H., Shang, W., Adams, B., Sayagh, M., & Hassan, A. E. (2020). A qualitative study of the benefits and costs of logging from developers' perspectives. *IEEE Transactions on Software Engineering*, *47*(12), 2858–2873.

Lucassen, G., Dalpiaz, F., Werf, J. M. E. M. V. D., & Brinkkemper, S. (2016). The Use and Effectiveness of User Stories in Practice. In M. Daneva & O. Pastor (Eds), *Requirements Engineering: Foundation for Software Quality* (Vol. 9619, pp. 205–222). Springer International Publishing. https://doi.org/10.1007/978-3-319-30282-9_14

Lung, C. F., & Van Der Bent, J. F. (2026). *Report on the definition of the browser extension* (D10.2; AI4Debunk).

Mavin, A., Wilkinson, P., Teufl, S., Femmer, H., Eckhardt, J., & Mund, J. (2017). Does goal-oriented requirements engineering achieve its goal? *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 174–183.

McGraw, G. (2006). *Software Security: Building Security in*. Addison-Wesley.

Mikkonen, T., Pautasso, C., Systa, K., & Taivalsaari, A. (2022). Cargo-Cult Containerization: A Critical View of Containers in Modern Software Development. *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 93–98. https://doi.org/10.1109/SOSE55356.2022.00017

Mitchell, B. S. (2023). Cloud Native Software Engineering. *arXiv Preprint arXiv:2307.01045*.

Muzumdar, P., Bhosale, A., Basyal, G. P., & Kurian, G. (2024). Navigating the Docker Ecosystem: A Comprehensive Taxonomy and Survey. *Asian Journal of Research in Computer Science*, *17*(1), 42–61. https://doi.org/10.9734/ajrcos/2024/v17i1411

Newman, S. (2015). *Building microservices: Designing fine-grained systems* (First edition). O'Reilly Media.

Olechowski, A. L., Eppinger, S. D., Joglekar, N., & Tomaschek, K. (2020). Technology readiness levels: Shortcomings and improvement opportunities. *Systems Engineering*, *23*(4), 395–408. https://doi.org/10.1002/sys.21533

Rani, P., Zellweger, J., Kousadianos, V., Cruz, L., Kehrer, T., & Bacchelli, A. (2024). Energy Patterns for Web: An Exploratory Study. *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Society*, 12–22. https://doi.org/10.1145/3639475.3640110

Sallou, J., Durieux, T., & Panichella, A. (2024). Breaking the Silence: The Threats of Using LLMs in Software Engineering. *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results, ICSE-NIER'24*, 102–106. https://doi.org/10.1145/3639476.3639764

Sauser, B., Gove, R., Forbes, E., & Ramirez-Marquez, J. E. (2010). Integration maturity metrics: Development of an integration readiness level. *Information Knowledge Systems Management*, *9*(1), 17–46. https://doi.org/10.3233/IKS-2010-0133

Schmitz, P.-E. (2013). The European Union Public Licence (EUPL). *International Free and Open Source Software Law Review*, *5*(2), 121–136. https://doi.org/10.5033/ifosslr.v5i2.91

Shalloway, A., & Trott, J. (2002). *Design Patterns Explained: A New Perspective on Object-oriented Design*. Addison-Wesley. https://books.google.nl/books?id=JPOaP7cyk6wC

Shi, J., Yang, Z., & Lo, D. (2025). Efficient and Green Large Language Models for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Transactions on Software Engineering and Methodology*, *34*(5), 1–22. https://doi.org/10.1145/3708525

Turnbull, J. (2018). *Monitoring with Prometheus*. Turnbull Press.

Van Vliet, H. (2008). *Software engineering: Principles and practice* (Vol. 13). John Wiley & Sons Hoboken, NJ.

Vernon, V. (2013). *Implementing domain-driven design*. Addison-Wesley.

Wahed, M. A., Alzboon, M. S., Alqaraleh, M., Ayman, J., Al-Batah, M., & Bader, A. F. (2024). Automating web data collection: Challenges, solutions, and python-based strategies for effective web scraping. *2024 7th International Conference on Internet Applications, Protocols, and Services (NETAPPS)*, 1–6.

Yousuf, B., Qureshi, M. A., Spillane, B., Munnelly, G., Carroll, O., Runswick, M., Park, K., Culloty, E., Conlan, O., & Suiter, J. (2021). *PROVENANCE: An Intermediary-Free Solution for Digital Content Verification* (arXiv:2111.08791). arXiv. https://doi.org/10.48550/arXiv.2111.08791

Zarour, M., Alzabut, H., & Al-Sarayreh, K. T. (2025). MLOps best practices, challenges and maturity models: A systematic literature review. *Information and Software Technology*, *183*, 107733. https://doi.org/10.1016/j.infsof.2025.107733

Zerouali, A., Opdebeeck, R., & De Roover, C. (2023). Helm Charts for Kubernetes Applications: Evolution, Outdatedness and Security Risks. *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, 523–533. https://doi.org/10.1109/MSR59073.2023.00078

Funded by the European Union

A RFC: WE NEED (MORE) REPEATABLE BUILDS

The text below was originally distributed to technical partners from WP6–11 on 24 September 2025. It is intended as a practical guide for partners developing backing services that need to be integrated into the debunking API. The text has been edited slightly to increase its suitability for public dissemination.

Target audience	Technical partners who develop software that is hosted by HU
Assumed knowledge	Python (intermediate), Docker (basic), GitHub Actions (basic)

HU University of Applied Sciences Utrecht (HU) is responsible for the development of the debunking API, which integrates various Python-based software components by consortium partners (WP6–9).

In an ideal world, consortium partners would develop software components and HU would be able to integrate software components simply by linking to them. Sadly, we do not live in an ideal world. Software that is written today may no longer be buildable in a year, a month, or even just a week. Some software only builds or runs correctly on machines that meet a very specific (and unknown) set of requirements. This poses significant challenges for HU, as the project's system integrator.

This document provides recommendations that aim to get our consortium closer to that ideal world in which everything "just works".

We ask that you follow the four recommendations below when developing your software. The first two recommendations (sections A.1 and A.2) must be done by you. The other recommendations (sections A.3 and A.4) can be done either by you or us, depending on how comfortable you are with Docker and GitHub Actions. Let us know what you prefer.

A.1 MAKE DEPENDENCIES EXPLICIT

Why we need this
Unless you declare all your dependencies, someone else may not be able to run your code at all without investing a significant amount of effort.

Software written in Python typically has three types of dependencies:

1. The Python runtime, e.g., Python 3.12, Python 3.13. This dictates which language features you can use. Modern setups often also use virtual environments[22].

[22] https://docs.python.org/3/library/venv.html

Funded by the European Union

2. Python dependencies, e.g., `numpy`, `scikit-learn`. These dependencies are available via the Python Package Index[23] (PyPI).
3. System dependencies, e.g., `build-essential`, `ffmpeg`. These dependencies are installed via your system's package manager (usually `apt`).

Python projects commonly aim to solve the second type of dependency using `pip`[24] along with `requirements.txt`[25] files that define which Python dependencies need to be installed. Pip is the *de facto* standard package installer in the Python ecosystem. However, it suffers from two major issues:

- Modern package installers distinguish between manifest files and lockfiles. Manifest files are a short "wishlist" of packages your software logically depends on ("I need a computer with a GPU"). Lockfiles declare exactly which package versions you used to run the software on your machine ("I used a computer with an AMD Ryzen Threadripper 7980X, 128GB DDR5-6400 memory, an NVIDIA GeForce RTX 5090 graphics card, etc."). Pip mixes these two concepts. This makes it harder to reliably restore environments and upgrade packages.
- Pip is prone to installing incompatible packages side by side. This may happen when packages are installed in the "wrong" order.

To mitigate these issues, we request that you use a modern package installer whenever possible, such as uv[26]. Alternatives such as `conda`[27] can possibly be used as well.

A.1.1 WORKING WITH UV

Before you start, make sure that uv is installed. You can do this using the standalone installer[28] or via `pip install uv`.

To create a new project, run the command in Listing 1[29]. This will create a basic `pyproject.toml` (the manifest file) and a `main.py`. If you already have an existing `requirements.txt`, run this command in an empty directory and copy the generated files to your existing project.

```
$ uv init example-app
```

LISTING 1: CREATING A NEW PYTHON APPLICATION WITH UV

[23] https://pypi.org/
[24] https://pip.pypa.io/en/stable/
[25] https://pip.pypa.io/en/stable/reference/requirements-file-format/
[26] https://docs.astral.sh/uv/
[27] https://docs.conda.io/projects/conda
[28] https://docs.astral.sh/uv/getting-started/installation/
[29] The dollar sign is not part of the command, but indicates that the command should be run as a regular (non-root) user

Funded by the European Union

Enter the newly created directory – in this case `example-app` – and run the command in Listing 2. This is the uv equivalent of `pip install -r requirements.txt`. Running the command for the first time will create a `uv.lock` (the lockfile) and a `.python-version` that respectively define the exact set of dependencies and the required Python version. Make sure you commit all files to Git.

```
$ uv sync
```

LISTING 2: INSTALLING REQUIREMENTS FROM A LOCKFILE WITH UV

To add a new Python package to your project, run the command in Listing 3. This will also install all transitive[30] dependencies. To remove a Python package, run the command in Listing 4. This will also remove all transitive dependencies that are not required by other packages. Finally, to run a Python script using the Python environment for your project use `uv run`, as shown in Listing 5.

```
$ uv add numpy
```

LISTING 3: ADDING A DEPENDENCY WITH UV

```
$ uv remove numpy
```

LISTING 4: REMOVING A DEPENDENCY WITH UV

```
$ uv run main.py
  Hello from example-app!
$ uv run src/foo/bar/baz.py # You can also run deeply nested files
```

LISTING 5: RUNNING A PYTHON SCRIPT WITH UV

A.2 MAKE YOUR SOFTWARE CONFIGURABLE

Why we need this

Run-time configurability gives your users more freedom to decide how they want to use your application.

To make it easier to run your software on other peoples' machines, avoid making assumptions about the environment in which they are run. Try to avoid hardcoding paths to files and folders. Instead, allow the user to provide these paths to your software so that your software can be run without modifying the source code.

[30] Indirect dependencies, i.e., your dependency's dependencies.

A.2.1 MODELS AND OTHER TYPES OF FILES

If your software relies on data or files that are not part of your source code, make sure that they can be set or overridden on application start (Listing 6). This makes it possible to easily change the behaviour of your software without having to rebuild the Docker image (see section A.3).

```
import argparse

parser = argparse.ArgumentParser(description='Some helpful text about the app')
parser.add_argument('--model', help='Name of model', default='llama3-70b-8192')
parser.add_argument('--path', help='Path to model', default='/ex/am/ple.onnx')

args = parser.parse_args()

print(args.model, args.path)
```

LISTING 6: CONFIGURING AN APPLICATION USING ENVIRONMENT VARIABLES

A.2.2 API KEYS AND SECRET TOKENS

Secrets such as passwords and API keys should not be passed directly via command-line arguments but instead be obtained from the environment (Listing 7).

```
import os

value = os.getenv('EXAMPLE_API_KEY')

# Easier for users, but strongly discouraged from security perspective
value_with_fallback = os.getenv('EXAMPLE_API_KEY') or '3xampl3-k3y'
```

LISTING 7: LOADING A SECRET FROM THE ENVIRONMENT

Listing 8 shows how to run a script with a specific environment variable in Linux. The same is also possible in Windows PowerShell[31] (Listing 9).

```
$  EXAMPLE_API_KEY=1234567890abcdef1234567890abcdef uv run example.py
```

LISTING 8: INVOKING A PYTHON SCRIPT WITH AN ENVIRONMENT VARIABLE IN *NIX

```
$env:EXAMPLE_API_KEY=1234567890abcdef1234567890abcdef; uv run example.py
```

LISTING 9: INVOKING A PYTHON SCRIPT WITH AN ENVIRONMENT VARIABLE USING WINDOWS POWERSHELL

[31] In this case the dollar sign is part of the command!

Funded by the European Union

A.3 CONTAINERISE (“DOCKERISE”) YOUR APPLICATION

Why we need this

Docker can be used to “package” your code so that it can be easily run on any machine, even if that machine does not have Python installed.

Docker is a technology that lets you package your code along with all its dependencies into a portable box that we call an “image”. Images can be published on the internet. Others can then easily download and run these images in an isolated manner, whether that is on a laptop or in a specialised container system distributed across multiple servers and geographic locations.

To support Docker, provide a `Dockerfile` in the root of your project that specifies the operating system in which your application will run and how its dependencies will be installed. Listing 10 shows a minimal example of a `Dockerfile`.

```
FROM python:3.13-slim

WORKDIR /app

# This copies all files in the current directory to the container image’s
# WORKDIR. The “current directory” is the directory on your computer that
# contains the Dockerfile.
COPY . .

# “RUN” is used for all commands that must have already been run before a user
# runs the image. These commands are run during image builds and typically
# include the installation of packages via apt and/or your Python package
# installer.
#
# If your model file isn’t very large (a few hundred MB at most), this may be a
# good place to download it. Otherwise provide a script (Python or shell) in
# your project that can be used to download it manually.
RUN uv sync

# “CMD” is used for a single command that will be used when the image is run.
# Note that main.py will be located at /app/main.py because we set the WORKDIR
# to /app.
CMD python main.py
```

LISTING 10: MINIMAL EXAMPLE OF A DOCKERFILE FOR UV-BASED PYTHON APPS

A.4 AUTOMATE TESTING

Why we need this

By automatically testing Docker images every time they are built, we can be confident that new versions work as expected and are safe to be released to the public (or our hosting environment).

Funded by the European Union

To ensure that the Docker image works correctly, it needs to be built and tested. We can automate these steps using a CI/CD pipeline, for example by executing a GitHub Actions workflow[32] that runs automatically after every commit. After completion, a green checkmark or red cross will show up on GitHub next to the commit, depending on whether the tests have run successfully.

What the workflow does is entirely up to you. You can go all out using a test framework such as `pytest`[33] or keep it simple by building an image and attempting to run it.

[32] https://docs.github.com/en/actions/how-tos/write-workflows
[33] https://docs.pytest.org/en/stable/

Funded by the European Union

Review Sheet of Deliverable/ Milestone Report
D10.1 Deliverable Title

Editor(s):	Franc van der Bent (HU) Chun Fei Lung (HU)
Responsible Partner:	Hogeschool Utrecht (HU University of Applied Sciences Utrecht)
Status-Version:	Final – v1
Date:	09/02/2026
Distribution level (CO, PU):	Public
Reviewer (Name/Organization)	Viktoriya Dimova, Ana Rita Alves (F6S)
Review date	23/02/2026

Disclaimer: This assessment reflects only the author's views and the European Commission is not responsible for any use that may be made of the information contained therein.

Funded by the European Union

Mark with X the corresponding column:

Y= yes	N= no	N = not applicable

ELEMENT TO REVIEW	Y	N	NA	COMMENTS
FORMAT: Does the document … ?				
…include editors, deliverable name, version number, dissemination level, date, and status?	X			
…contain a license (in case of public deliverables)?	X			
…include the names of contributors and reviewers?	X			
….have a version table consistent with the document's revision?	X			
… contain an updated table of contents?	X			
… contain a list of figures consistent with the document's content?	X			
… contain a list of tables consistent with the document's content?	X			
… contain a list of terms and abbreviations?	X			
… contain an Executive Summary?	X			
… contain a Conclusions section?	X			
… contain a List of References (Bibliography) in the adequate format, if relevant?	X			
… use the fonts and sections defined in the official template?	X			
… use correct spelling and grammar?	X			
… conform to length guidelines (50 pages maximum (plus Executive Summary and annexes)	X			
… conform to guidelines regarding Annexes (inclusion of complementary information)	X			
… present consistency along the whole document in terms of English quality/style? (to avoid accidental usage of copy-pasted text)	X			
About the content…				
ELEMENT TO REVIEW	**Y**	**N**	**NA**	**COMMENTS**
Is the overall style of the deliverable correctly organized and presented in a logical order?	X			
Is the Executive Summary self-contained, following the guidelines and does it include the main conclusions of the document?	X			

Funded by the European Union

ELEMENT TO REVIEW	Y	N	NA	COMMENTS
Is the body of the deliverable (technique, methodology results, discussion) well enough explained?	X			
Are the contents of the document treated with the required depth?	X			
Does the document need additional sections to be considered complete?		X		
Are there any sections in the document that should be removed?		X		
Are all references in the document included in the references list?	X			
Have you noticed any text in the document not well referenced? (copy and paste of text/picture without including the reference in the reference list)		X		
SOCIAL and TECHNICAL RESEARCH WPs (WP4, 5, 12, 13, 14)				
ELEMENT TO REVIEW	**Y**	**N**	**NA**	**COMMENTS**
Is the deliverable sufficiently innovative?			X	
Does the document present technical soundness and its methods are correctly explained?			X	
What do you think is the strongest aspect of the deliverable?			X	
What do you think is the weakest aspect of the deliverable?			X	
Please perform a brief evaluation and/or validation of the results, if applicable.			X	
AI AND TECNOLOGICAL WPS (WP6 – WP11)				
ELEMENT TO REVIEW	**Y**	**N**	**NA**	**COMMENTS**
Does the document present technical soundness and the methods are correctly explained?	X			
What do you think is the strongest aspect of the deliverable?				The deliverable is comprehensive, well structured and practical, technical guidance for creating a robust and scalable API, including modern architecture. The content is well researched, with mentions of

Funded by the European Union

ELEMENT TO REVIEW	Y	N	NA	COMMENTS
				varied recent publications on the topics.
What do you think is the weakest aspect of the deliverable?				Limitations and future work sections could have been more detailed to include action points to be addressed by AI4Debunk.
Please perform a brief evaluation and/or validation of the results, if applicable.				The deliverable is a strong and comprehensive document that provides a solid foundation for the AI4Debunk project's core technical component.
DISSEMINATION AND EXPLOITATION WPs (WP15 – WP17)				
ELEMENT TO REVIEW	**Y**	**N**	**NA**	**COMMENTS**
Does the document present a consistent outreach and exploitation strategy?			X	
Are the methods and means correctly explained?			X	
What do you think is the strongest aspect of the deliverable?			X	
What do you think is the weakest aspect of the deliverable?			X	
Please perform a brief evaluation and/or validation of the results, if applicable.			X	
DISSEMINATION AND EXPLOITATION WPs (WP18)				
ELEMENT TO REVIEW	**Y**	**N**	**NA**	**COMMENTS**
Does the document present the main ethical aspects regarding the use of methods and human involvement?			X	
What do you think is the strongest aspect of the deliverable?			X	
What do you think is the weakest aspect of the deliverable?			X	
Please perform a brief evaluation and/or validation of the results, if applicable.			X	

SUGGESTED IMPROVEMENTS

PAGE	SECTION	SUGGESTED IMPROVEMENT

Funded by the European Union

CONCLUSION

Mark with X the corresponding line.

X	Document accepted, no changes required.
	Document accepted, changes required.
	Document not accepted, it must be reviewed after changes are implemented.

Please rank this document globally on a scale of 1-5 (1 = poor, 5= excellent) – using a half point scale.
Mark with X the corresponding grade.

Document grade	1	1.5	2	2.5	3	3.5	4	4.5	5
									X