



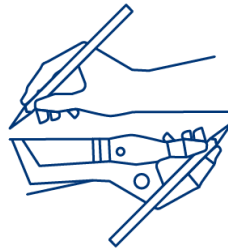
Funded by
the European Union

Project: 101094364 — ITHACA — HORIZON-CL2-2022-DEMOCRACY-01

EUROPEAN RESEARCH EXECUTIVE AGENCY (REA)

REA.C – Future Society

C.1 – Inclusive Society



ITHACA
AI To Enhance Civic Participation

ITHACA

artificial Intelligence To enHance Civic pArticipation

ITHACA_D3.2_Technical requirements, architecture design and 1st release - prototype

Work Package: WP3 – ITHACA platform design and development

Authors:	Konnektable, SIMAVI, UPAT
Status:	Final
Due Date:	31/01/2025
Version:	1.00
Submission Date:	24/06/2025
Dissemination Level:	R - Public

Disclaimer:

This document is issued within the frame and for the purpose of the ITHACA project. This project has received funding from the European Union's Horizon Europe Framework Programme under Grant Agreement No. 101094364. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

This document and its content are the property of the ITHACA Consortium. All rights relevant to this document are determined by the applicable laws. Access to this document does not grant any right or license on the document or its contents. This document or its contents are not to be used or treated in any manner inconsistent with the rights or interests of the ITHACA Consortium or the Partners detriment and are not to be disclosed externally without prior written consent from the ITHACA Partners. Each ITHACA Partner may use this document in conformity with the ITHACA Consortium Grant Agreement provisions.

(*) Dissemination level. - Public — fully open (automatically posted online)

Sensitive — limited under the conditions of the Grant Agreement

EU classified — RESTREINT-UE/EU-RESTRICTED, CONFIDENTIEL-UE/EU-CONFIDENTIAL, SECRET-UE/EU-SECRET under Decision 2015/444

ITHACA Project Profile

Grant Agreement No.: 101094364

Acronym:	ITHACA
Title:	artificial Intelligence To enHAnce Civic pArticipation
URL:	TBA
Start Date:	01/01/2023
Duration:	36 months

Partners

Short Name	Legal Name	Country
KT	KONNEKT ABLE TECHNOLOGIES LIMITED	IE
CERTH	ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS	EL
UPAT	PANEPISTIMIO PATRON	EL
RtF	RAISING THE FLOOR	BE
SnP	STAMADIANOS KAI SYNETAIROI DIKIGORIKI ETAIREIA	EL
UniGraz	UNIVERSITAET GRAZ	AT
MNLТ	MNLТ INNOVATIONS IKE	EL
SIMAVI	SOFTWARE IMAGINATION & VISION SRL	RO
PEDAL	PEDAL CONSULTING SRO	SK
BMA	AGENTIA METROPOLITANA PENTRU DEZVOLTARE DURABILA BRASOV ASOCIATIA	RO
MARTIN	MESTO MARTIN	SK

D3.2_Technical requirements, architecture design and 1st release - prototype

DOCUMENT HISTORY

VERSION	DATE	CHANGES	RESPONSIBLE PARTNER
0.1	14/11/2024	ToC	Adrian Dragota (SIMAVI)
0.2	14/01/2025	Content creation	Adrian Dragota (SIMAVI)
0.3	15/03/2025	Adaptations	Evangelos Rigas (KT), Epameinondas Koutavelis (KT)
0.4	15/06/2025	Additions	Iliana Loi (UPAT), Adrian Dragota (SIMAVI), Evangelos Rigas (KT)
1.0	24/06/2025	Finalisation	Epameinondas Koutavelis (KT), Evangelos Rigas (KT)

Table of Contents

1.	Introduction.....	7
1.1	Objectives of the Deliverable.....	7
1.2	Structure of the document.....	7
2.	Operational Prototype – Overview of the Integrated Platform.....	8
3.	Deployment Architecture and Infrastructure	11
3.1	Hardware and software infrastructure	11
4.	Demonstration of the operational prototype.....	14
4.1	Components	14
4.1.1	T1 - Evidence extraction from citizen’s discussions on the informal web	14
4.1.2	T2 - Topic Extraction	15
4.1.3	T3 - Toxicity Detection Tool / Automated moderation	18
4.1.4	T4 – Ontology	19
4.1.5	T5 - Gamification Incentive Mechanism.....	19
4.1.6	T6 - Public AI Register	21
4.1.7	T7 - Argument visualization.....	21
4.1.8	T8 – AI Fairness	23
4.1.9	T9 - PPML tool	24
4.1.10	T10 – AI Cybersecurity tool	29
4.1.11	T11 – Simplification Tool	30
4.1.12	T12 – Summarization Tool	33
4.1.13	T13 – Translation Tool	38

D3.2_Technical requirements, architecture design and 1st release - prototype

4.1.14	T14 – Chatbot	39
5.	Demonstrator Operation and Key Functionalities	40
5.1	Overview	40
5.2	Main actions.....	40
5.1.1	Login/register	40
5.1.2	The main dashboard	42
5.1.3	Forum page.....	42
5.1.4	Chat page.....	45
5.1.5	Reported posts	45
5.1.6	Login Logs page.....	46
5.1.7	Feedback page	46
5.1.8	Profile page.....	47
6.	Challenges and Limitations	47
6.1	Technical Challenges.....	47
7.	Conclusions and Next Steps.....	48
8.	References	50

List of Figures

Figure 1 Architectural layers	11
Figure 2 Hardware characteristics	11
Figure 3 Hardware characteristics 2	12
Figure 4 VPN Configuration.....	12
Figure 5 Operating system: Ubuntu 23.10	13
Figure 6 Operating System architecture.....	14
Figure 7 The Leaderboard Screen.....	20
Figure 8 The list of the currently Active Missions for the user to complete	20
Figure 9 An example of the screen appearing in the ITHACA user interface. Upon selecting a public issue, a list of relevant proposals will appear.	22
Figure 10 An example of the screen appearing in the ITHACA user interface when selecting a proposal for a public issue.....	22
Figure 11 Login / Register	40
Figure 12 Registration screen	41
Figure 13 Main dashboard.....	42
Figure 14 Forum page.....	42
Figure 15 Creation of a new proposal	43
Figure 16 Toxicity filter.....	44
Figure 17 Reply, Summarize, Simplify, Translate buttons	44
Figure 18 Chat Page	45
Figure 19 Reported posts.....	45
Figure 20 Login logs page.....	46
Figure 21 Feedback page	46
Figure 22 Profile page.....	47

EXECUTIVE SUMMARY

The **D3.2** deliverable provides a comprehensive overview of the technical requirements, architectural design, and initial prototype implementation of the ITHACA platform. It serves as a foundational document for aligning the system's design with its strategic goals, as defined in the ITHACA project funded under the Horizon Europe program.

The document outlines the following key aspects:

1. **Technical Requirements**

This section defines both functional and non-functional requirements, ensuring the platform meets performance, scalability, and security standards. These requirements are derived from prior deliverables and aligned with the project's overarching objectives.

2. **System Architecture**

The ITHACA platform's architecture is designed to support modularity and interoperability, enabling seamless integration of various components. The architecture includes:

- o **Data Collection Modules:** For evidence extraction and data acquisition from multiple sources.
- o **Processing and Analysis Tools:** Leveraging advanced technologies such as NLP for topic extraction, toxicity detection, and gamification incentive mechanisms.
- o **User Interaction Components:** Including visualization tools and a chatbot for enhanced user engagement.

3. **Prototype Implementation**

The initial prototype demonstrates the integration of core components, including tools for data collection, processing, and visualization. The deliverable provides an overview of the tools and technologies utilized, such as Docker for containerization and Key cloak for authentication.

4. **Testing and Validation**

The prototype undergoes rigorous testing to validate its functionality and ensure compliance with technical requirements. Test scenarios and results are discussed to highlight the system's capabilities and areas for improvement.

5. **Challenges and Lessons Learned**

The development and integration process revealed key challenges, such as system interoperability and data privacy concerns. These insights inform the project's ongoing refinement and future development phases.

The deliverable underscores the collaborative efforts of the consortium in delivering a robust and scalable platform. It marks a significant milestone in the ITHACA project, providing the groundwork for subsequent iterations and enhancements.

1. Introduction

1.1 Objectives of the Deliverable

This deliverable aims to provide a detailed report on the technical requirements, architectural design, and first operational prototype of the ITHACA platform. It aligns with the project's goal of enhancing civic participation through innovative AI-driven tools.

Specifically, this document seeks to:

1. Define the **technical and functional requirements** for the ITHACA platform, including aspects of security, scalability, and interoperability.
2. Present the **architectural design** that supports modularity, service-based principles, and adherence to the European Interoperability Framework (EIF).
3. Demonstrate the **integration and deployment** of the platform's early prototypes, including user-friendly interfaces, security services, and trustful AI tools.
4. Highlight the platform's core components, including data collection, NLP-based topic extraction, gamification, and argument visualization.
5. Document the testing, validation, and early user feedback to ensure alignment with technical and user-centered objectives.
6. Identify challenges encountered during development, particularly in areas such as system interoperability and data privacy and propose mitigation strategies.
7. Provide insights into the next steps for refining and scaling the platform to meet the project's overall objectives.

1.2 Structure of the document

The document is organized as follows:

1. **Introduction:**
 - o Presents the objectives and scope of the deliverable.
 - o Provides an overview of the structure and methodology.
2. **Operational Prototype – Overview of the Integrated Platform:**
 - o Details the system architecture and logical design.
 - o Outlines integration and security requirements.
3. **Deployment Architecture and Infrastructure:**
 - o Describes the physical and logical deployment strategies.
 - o Highlights the hardware and software infrastructure used.
4. **Demonstration of the Operational Prototype:**

- o Explains the functionalities of key components (e.g., data extraction, topic analysis, chatbot, gamification).
 - o Includes screenshots and step-by-step guides for usability testing.
5. **Demonstrator Operation and Key Functionalities:**
- o Provides a walkthrough of the platform's main actions, from user login to interaction with core components.
6. **Challenges and Limitations:**
- o Discusses technical challenges, such as scalability and security issues.
 - o Identifies limitations encountered during integration and testing.
7. **Conclusions and Next Steps:**
- o Summarizes findings and highlights recommendations for the next phases of development.
8. **References:**
- o Lists all related deliverables and supporting documents.

2. Operational Prototype – Overview of the Integrated Platform

The integrated ITHACA platform is hosted in a cloud environment utilizing a **Linux-based machine**. The platform deployment architecture includes two main components: **standalone services** and **Docker containers**. Docker is employed for containerization due to its open-source nature and robust ecosystem for managing lightweight, standalone executable packages. These containers encapsulate everything needed to run specific modules, ensuring portability and consistency across environments. The architecture also incorporates **Apache Kafka** for message exchange between components, **Keycloak** for user authentication and authorization, and **MongoDB** for document-based storage.

Platform Services

The platform comprises the following key services, deployed and orchestrated through Docker Compose:

1. **Spring Boot Backend:**
 - o **Service Name:** springboot-app
 - o The backend is a Java-based Spring Boot application that connects to a PostgreSQL database and provides core backend functionalities.
 - o It runs on port 8090, with environment variables configuring the database connection (SPRING_DATASOURCE_URL,

SPRING_DATASOURCE_USERNAME, SPRING_DATASOURCE_PASSWORD)
and activating the platform profile.

- o A volume is mounted to provide persistent storage for application-related resources (/home/ithaca/pictures).

2. PostgreSQL Database:

- o **Service Name:** postgres
- o Acts as the primary relational database for Keycloak and other relational data needs.
- o Runs on port 5432 and is configured with the necessary credentials (POSTGRES_USER, POSTGRES_PASSWORD, POSTGRES_DB) to store user data and platform configurations.
- o A named volume (postgres-data) ensures persistent data storage.

3. Frontend Application:

- o **Service Name:** ithaca-app
- o The front-end is an Angular-based application served using **NGINX**.
- o It utilizes self-signed certificates for secure HTTPS communication, with key and certificate files mounted from the host (selfsigned.crt, selfsigned.key, and dhparam.pem).
- o The front-end exposes ports 80 (HTTP) and 443 (HTTPS) and serves a pre-built Angular distribution from the dist folder.

4. MongoDB Database:

- o **Service Name:** mongo
- o MongoDB provides a document-oriented database solution, which is particularly useful for handling unstructured or semi-structured data in JSON-like formats.
- o Runs on port 27017 and is initialized with a default database (admin) and root credentials (root/password).
- o A named volume (mongo-data) ensures persistent storage of MongoDB data.
- o MongoDB's flexibility makes it suitable for storing dynamic content and large datasets such as user activity logs or session data.

Docker Configuration

The platform is configured using **Docker Compose** to orchestrate the following:

- **Service Dependencies:** The backend service (springboot-app) depends on the database services (postgres and mongo) for proper initialization.
- **Network:** All services are connected via a Docker-managed bridge network (app-network), enabling seamless communication between containers.

Build and Deployment Workflow

1. Backend (Spring Boot):

- o A Dockerfile is used to containerize the Spring Boot application.
- o The application JAR file (ITHACA-0.0.1-SNAPSHOT.jar) is copied into the container, and port 8090 is exposed for external access.
- o A Keycloak keystore (keycloak.p12) is included for secure integration with the authentication service.

2. Frontend (Angular):

- o The Angular application is built using Node.js (npm run build) to generate the production-ready dist folder.
- o The final build is then served via **NGINX**, with custom configurations defined in nginx.conf.

3. MongoDB:

- o A MongoDB container is initialized with persistent data storage, ensuring reliable handling of document-based data.

Rationale for Tools and Technologies

1. **Docker**: Chosen for its simplicity and widespread adoption in managing containerized services. Docker ensures consistency across development, testing, and production environments.
2. **NGINX**: Used for serving the frontend application and acting as a reverse proxy for secure and efficient web serving.
3. **PostgreSQL**: Selected as the relational database for its robustness, scalability, and strong community support.
4. **MongoDB**: Provides a flexible, document-based storage solution for handling non-relational data.
5. **Kafka**: Ensures reliable and high-performance messaging between platform components.
6. **Keycloak**: Provides centralized identity and access management for seamless and secure user authentication.

Development and Continuous Integration

- The deployment process adheres to **CI/CD principles**, enabling automated builds and testing of platform components using tools like Jenkins. This ensures efficient delivery cycles and high-quality releases.

System Architecture Reference

The proposed architecture is detailed in D3.1: Technical requirements, architecture design and 1st release - report. For completeness, the logical architecture diagram from D3.1 has been extracted and included in this document.

In the operational prototype, the architectural layers consist of the following components:

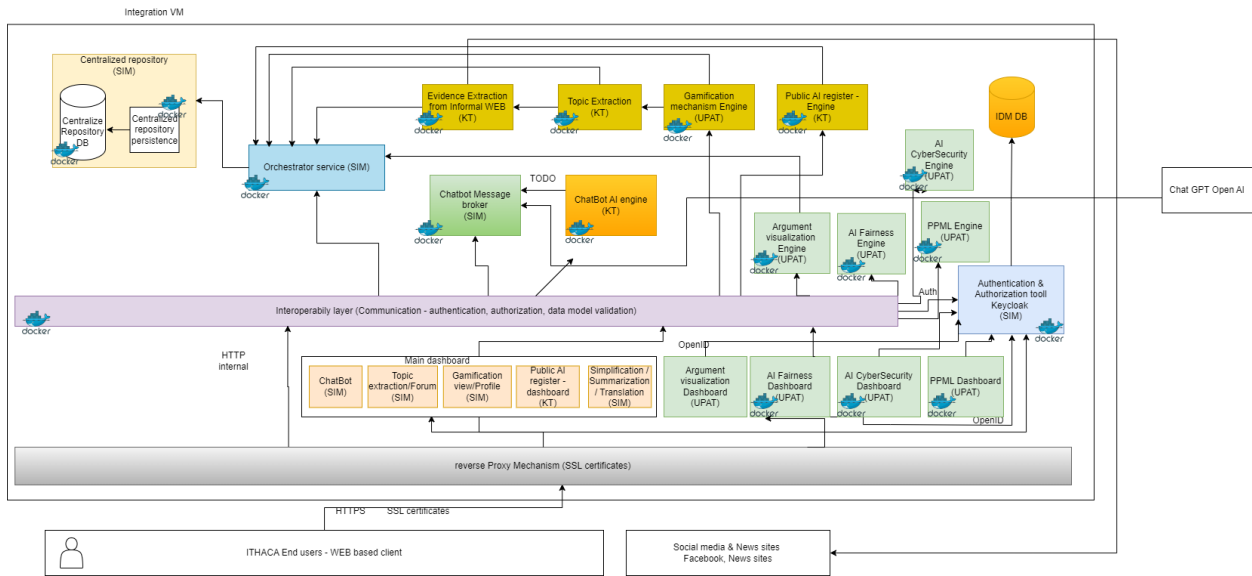


Figure 1 Architectural layers

3. Deployment Architecture and Infrastructure

3.1 Hardware and software infrastructure

The hardware infrastructure is mapped on a SIMAVI instance. At the beginning of deployment an instance with the following characteristics is used:

- 16 CPU's
- 64GB RAM
- 500GB SSD

```

0[ 0.0%] 4[ 0.0%] 8[ 0.0%] 12[ 0.0%]
1[ 0.0%] 5[ 0.0%] 9[ 0.0%] 13[ 0.0%]
2[ 0.0%] 6[ 0.0%] 10[ 0.0%] 14[ 0.0%]
3[ 0.6%] 7[ 0.0%] 11[ 0.0%] 15[ 3.8%]
Mem[ ] |18.8G/62.8G| Tasks: 115, 1312 thr, 255 kthr; 1 running
Swp[ ] | 216K/8.00G| Load average: 0.08 0.02 0.01
                                     Uptime: 88 days, 22:55:27
Main 1/0
  
```

Figure 2 Hardware characteristics

```
Architecture:          x86_64
CPU op-mode(s):      32-bit, 64-bit
Address sizes:       45 bits physical, 48 bits virtual
Byte Order:          Little Endian
CPU(s):              16
On-line CPU(s) list: 0-15
Vendor ID:           GenuineIntel
BIOS Vendor ID:     GenuineIntel
Model name:          Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz
BIOS Model name:    Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz  CPU @ 1.9GHz
BIOS CPU family:    2
CPU family:          6
Model:               85
Thread(s) per core: 1
Core(s) per socket: 1
Socket(s):           16
Stepping:            7
BogoMIPS:            4190.15
```

Figure 3 Hardware characteristics 2

VPN Configuration

For technical partners to access the instance, vpn access is required.

The VPN solution chosen was FortiClient. It offers a 2-factor authentication, with a token received via sms after the connect button is pressed.

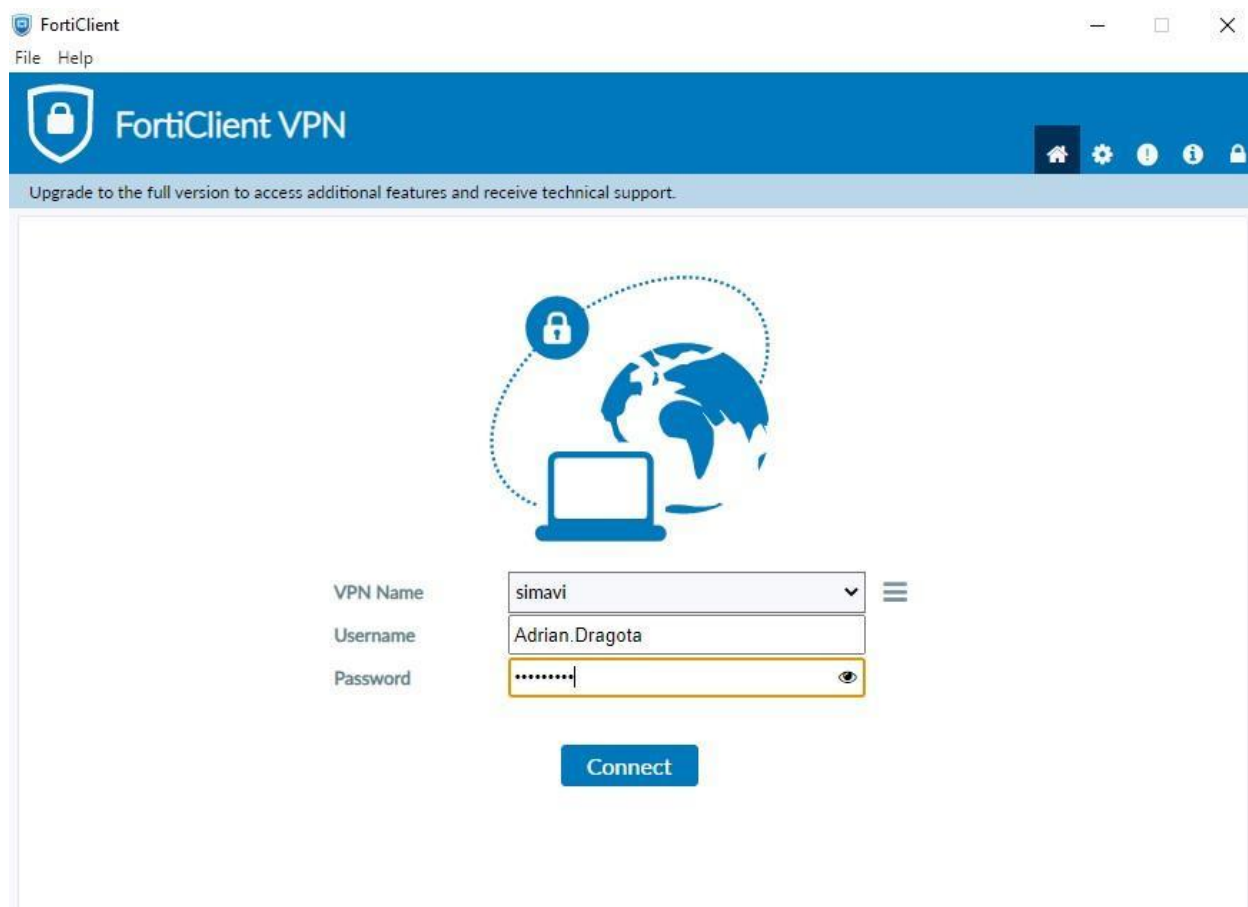


Figure 4 VPN Configuration

The IP behind the VPN is 10.222.30.234. Technical partners have access to SSH and any port that is needed for their containers.

In order for the front-end app to be reachable from the internet, a mapping was made as follows:

Angular front-end app: <https://ithaca.simavi.ro/> => https://10.222.30.234:443/

Keycloak: <https://keycloak.ithaca.simavi.ro/> => <https://10.222.30.234:8443/>

API: <https://ithaca.simavi.ro/api/combackend/> => <https://10.222.30.234:8090/>

Access to the API is done with a Bearer token, provided to the technical partners.

Software infrastructure

The following elements are considered for the software infrastructure:

The operating system is Ubuntu 23.10

```
root@ithaca:/home/ithaca# lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 23.10
Release:       23.10
Codename:      mantic
```

Figure 5 Operating system: Ubuntu 23.10

The placement of modules in the deliverables is explained below:

D3.2

- T1 - Evidence extraction from citizen's discussions on the informal web
- T2 - Topic Extraction
- T3 - Toxicity Detection Tool / Automated moderation
- T4 - Ontology
- T5 - Gamification incentive mechanism
- T6 - Public AI Register
- T7 - Argument Visualization
- T11 - Simplification Tool
- T12 - Summarization Tool
- T13 - Translation
- T14 - Chatbot

D5.1

- T8 - AI Fairness
- T9 - PPML Tool
- T10 - AI CyberSecurity Tool

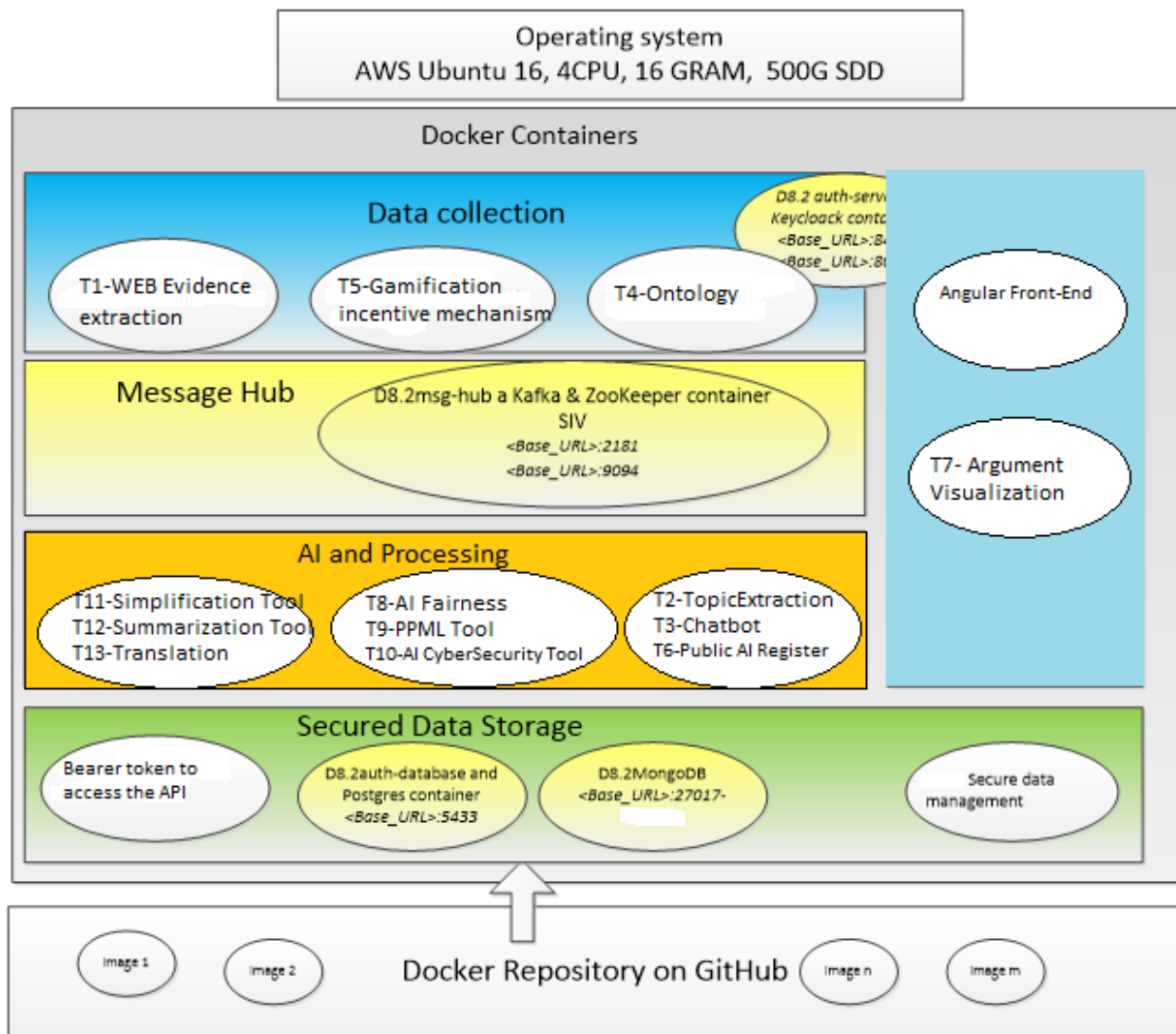


Figure 6 Operating System architecture

4. Demonstration of the operational prototype

4.1 Components

4.1.1 T1 - Evidence extraction from citizen's discussions on the informal web

Provider: KT

Work Package: WP3, T3.3 – Evidence extraction from citizen's discussions on the informal web

Short description of final module:

Collection of multilingual (Romanian and Slovak) content from the Web, including news sources from the two pilot cities and public social media content (e.g., social media pages of the pilot partners) to gain a deeper understanding of citizens' opinions with sentiment analysis by identifying how citizens feel about the different subjects of the discussion.

Installation and configuration:

The installation and configuration of the initial version of the Web crawler includes a Dockerfile used for building the respective docker image:

```
# Use an official Python runtime as a parent image
FROM python:3.6.9

# Set the working directory
WORKDIR /KT

...

sudo docker build -t kt/web_crawler:0.1 .
sudo docker run --name web_crawler -d --network host kt/web_crawler:0.1
```

Running (on Ubuntu):

From within the same directory as the above docker-compose file

```
sudo docker-compose up -d
```

Produced messages (topics):

N/A

Consumed Messages (topics):

N/A

4.1.2 T2 - Topic Extraction

Provider: KT

Work Package: WP3, T3.4 – Cognitive and social agents

Short description of final module:

NLP tools to cluster text posts by citizens on the ITHACA platform and the informal web (content search, selection, acquisition, categorization Furthermore, key phrase extraction techniques will be used to understand common topics and trends, such as the discussion topics.

Relevant code snippets:

Scenario A - Categorization through sequence classification:

```
def categorization():
    from transformers import AutoTokenizer, AutoModelForSequenceClassification

    texts = [ . . . ]
```

```

from transformers import MarianMTModel, MarianTokenizer

# Load Romanian to English model
model_name = "Helsinki-NLP/opus-mt-ro-en" # Replace with 'sk-en' for Slovak
tokenizer = MarianTokenizer.from_pretrained(model_name,
token=settings.HUGGING_TOKEN)
model = MarianMTModel.from_pretrained(model_name, token=settings.HUGGING_TOKEN)

for i in range(0,5):
    texts[i] = translate_text(texts[i], tokenizer, model)

from transformers import AutoModelForSequenceClassification, AutoTokenizer
from scipy.special import expit

model_name = "cardiffnlp/tweet-topic-large-multilingual"
model = AutoModelForSequenceClassification.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)

for text in texts:
    tokens = tokenizer(text, return_tensors='pt')
    output = model(**tokens)

    # Apply sigmoid to handle multi-label classification
    scores = expit(output.logits.detach().numpy()[0])

    # Threshold predictions
    threshold = 0.5
    predictions = (scores >= threshold).astype(int)

    # Map to topics
    topic_labels = model.config.id2label
    for idx, pred in enumerate(predictions):
        if pred:
            print(f"Predicted topic: {topic_labels[idx]}")

```

Scenario B - Clustering and keywords through a sentence transformer:

```

def categorization_cluster():
    from sentence_transformers import SentenceTransformer
    from transformers import MarianMTModel, MarianTokenizer

    # Load a multilingual embedding model

```

```

model = SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2')

texts = [ . . . ]

import unicodedata
for i in range(0,5):
    texts[i] = texts[i].replace("ț", "t").replace("ș", "s").replace("Ț",
"t").replace("Ș", "s")

model = SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2')
# Generate embeddings
embeddings = model.encode(texts)

from sklearn.cluster import KMeans

# Define number of topics (clusters)
num_topics = 3
kmeans = KMeans(n_clusters=num_topics, random_state=10)

# Fit and predict
clusters = kmeans.fit_predict(embeddings)

# Print results
for i, text in enumerate(texts):
    print(f"Text: {text} -> Topic: {clusters[i]}")

import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from ai_tools.stopwords import romanian_stopwords

# Find top texts for each cluster
for cluster_id in range(num_topics):
    cluster_texts = np.array(texts)[clusters == cluster_id]
    print(f"Cluster {cluster_id} Texts: {cluster_texts}")
    vectorizer = CountVectorizer(stop_words="english",
token_pattern=r'(?u)\b\w\w+\b', analyzer='word')
    word_counts = vectorizer.fit_transform(cluster_texts)
    common_words = vectorizer.get_feature_names_out()
    print(f"Cluster {cluster_id} Key Words: {common_words[:10]}")

```

4.1.3 T3 - Toxicity Detection Tool / Automated moderation

Provider: KT

Work Package: WP3, T3.4 – Cognitive and social agents

Short description of final module:

An automated moderator “bot” to assist with human-performed moderation, enabling the identification of inappropriate content submitted by users.

Relevant code snippets:

```
from transformers import (
    pipeline, AutoTokenizer, AutoModelForSequenceClassification)

def check_inappropriate_text(text):
    # Load tokenizer and model:
    model_name = "unitary/toxic-bert"

    tokenizer = AutoTokenizer.from_pretrained(model_name,
        token=settings.HUGGING_TOKEN)

    model = AutoModelForSequenceClassification.from_pretrained(model_name,
        token=settings.HUGGING_TOKEN)

    # Create a classification pipeline and evaluate input text:
    classifier = pipeline("text-classification", model=model, tokenizer=tokenizer)
    result = classifier(text)
    return result

class TextCheckView(views.APIView):
    def post(self, request, format=None):
        result = check_inappropriate_text(request.data['text'])
        inappropriate = False
        # Flag texts with toxicity score greater than half as inappropriate.
        # The threshold value can be adjusted according to needs.
        if result[0]['score'] > 0.5:
            inappropriate = True
        return Response({'inappropriate': inappropriate, 'score':
            result[0]['score']}, status=status.HTTP_200_OK)
```

4.1.4 T4 – Ontology

Provider: KT

Work Package: WP3, T3.5 – Ontology

Short description of final module:

Systematic organizing of the ontological knowledge and design of the ITHACA information space reusing/developing ontologies in the following fields: e-government public services. This task will carry out the integration of the ontology building and maintenance, semantic annotation, semantic indexing, search and retrieval, and semantic reasoner.

Development is set to commence in the next development phase.

4.1.5 T5 - Gamification Incentive Mechanism

Provider: UPAT

Work Package: WP3, T3.6 – Gamification incentive mechanism

Short description of final module:

The gamification engine is a system to reward users based on their participation with the platform. Its main goal is to enhance the quality of engagement of the end users by encouraging intrinsic motivation in terms of competence, relatedness, and activity.

The tool's main functionality, as described in deliverable D3.1, includes:

- The users being provided with missions, i.e., tasks to fulfill within the platform, which upon completion, they will be rewarded with:
 - Experience points helping the user to progress over their level.
 - Associated Badges (similar to trophies).
- Both missions and badges are categorized based on the type of intrinsic motivation they encourage (competence, relatedness, and activity) as well as their difficulty (e.g. bronze badges are awarded to easy missions, etc).
- The users will be ranked among other users based on the total experience points they collect from missions. Their name will be displayed in a leaderboard screen as the one in Figure 7.
- The users' level, total experience points, experience points to progress to the next level, as well as completed missions, obtained and available badges, will be displayed on their profile.

The engine's frontend has already been developed, while the implementation of the engine's backend is currently in progress. Figure 7 depicts the Leaderboards tab of the Profile section, where the top 4 users are displayed based on the total experience points they have obtained from missions.

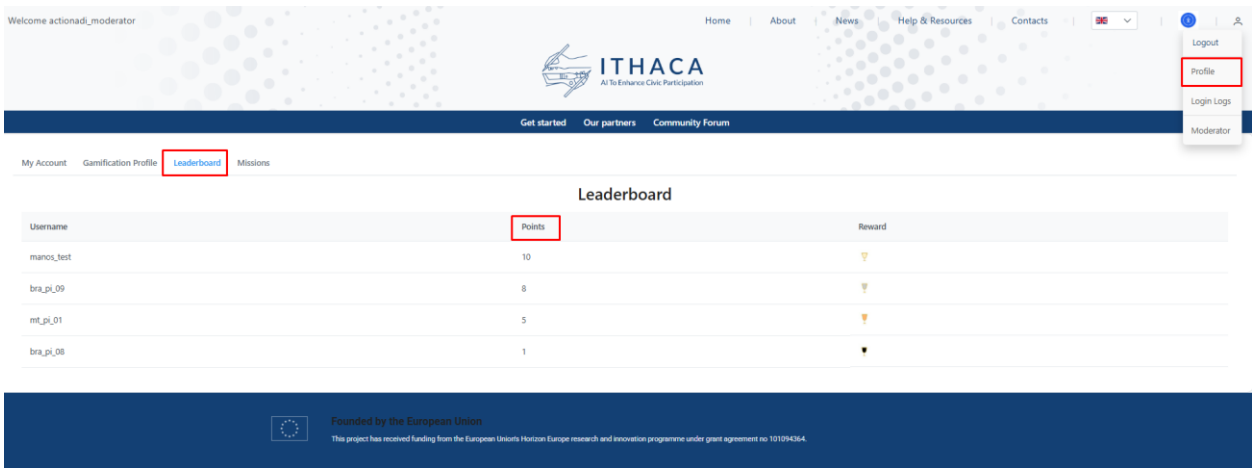


Figure 7 The Leaderboard Screen

Above

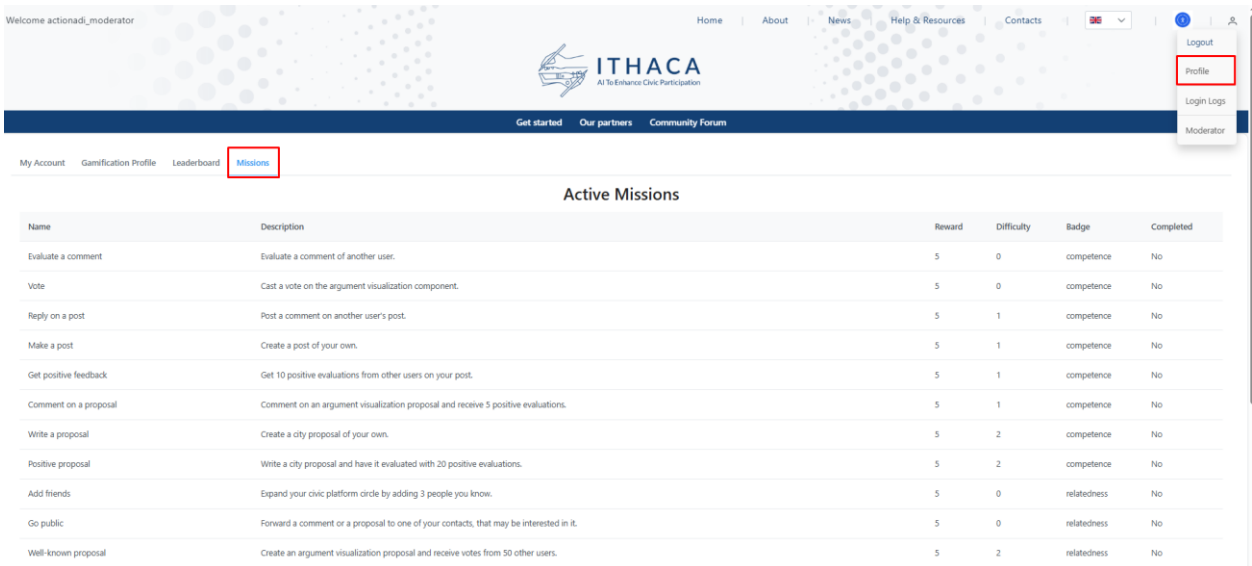


Figure 8 The list of the currently Active Missions for the user to complete

In Figure 8, the Active Missions tab of the Profile section is illustrated, where the current available missions are presented. The difficulty of each mission, along with its associated reward and badge to be obtained upon completion, are depicted next to the description of each mission. The completion status of each mission is also provided in the rightmost column.

4.1.6 T6 - Public AI Register

Provider: KT

Work Package: WP3, T3.7 – Public AI Register

Short description of final module:

Design and implementation of the webpages of the Public AI registers (in the pilot cities languages) and development of the content of the registers, providing information about the decisions and assumptions that were made in the process of developing, implementing, managing and ultimately dismantling algorithms used in the ITHACA platform.

Implementation is set to commence in the next development phase.

4.1.7 T7 - Argument visualization

Provider: UPAT

Work Package: WP3, T3.8 – Argument Visualization

Short description of final module:

The Argument Visualization component is designed to present the users'/citizens' suggestions and arguments regarding a civic issue, as well as a way to visualize their agreement or disagreement on these ideas using a pie chart.

Overall, as stated in D3.1, the Argument Visualization is a forum-like environment where the users/citizens:

- Can post their proposals regarding a civic issue. Proposals will provide ideas or solutions to public issues, posted by either citizens or users with elevated privileges (such as municipal users from cities testing the platform).
- Can vote for the best idea (proposal) to solve this issue.
- Can post their arguments in support of or against an existing proposal for a particular issue (state their agreement or disagreement).
- The citizen's arguments under each proposal can also be upvoted/downvoted.

A first version of the Argument Visualization tool has already been incorporated into the ITHACA platform. Figure 9, illustrates an example of a list of proposals for a civic issue (3 proposals can be seen above the pie chart). The citizens/users may post their arguments under each proposal in the form of comments, as well as vote for a proposal (like button). The pie chart visually represents the votes across proposals.

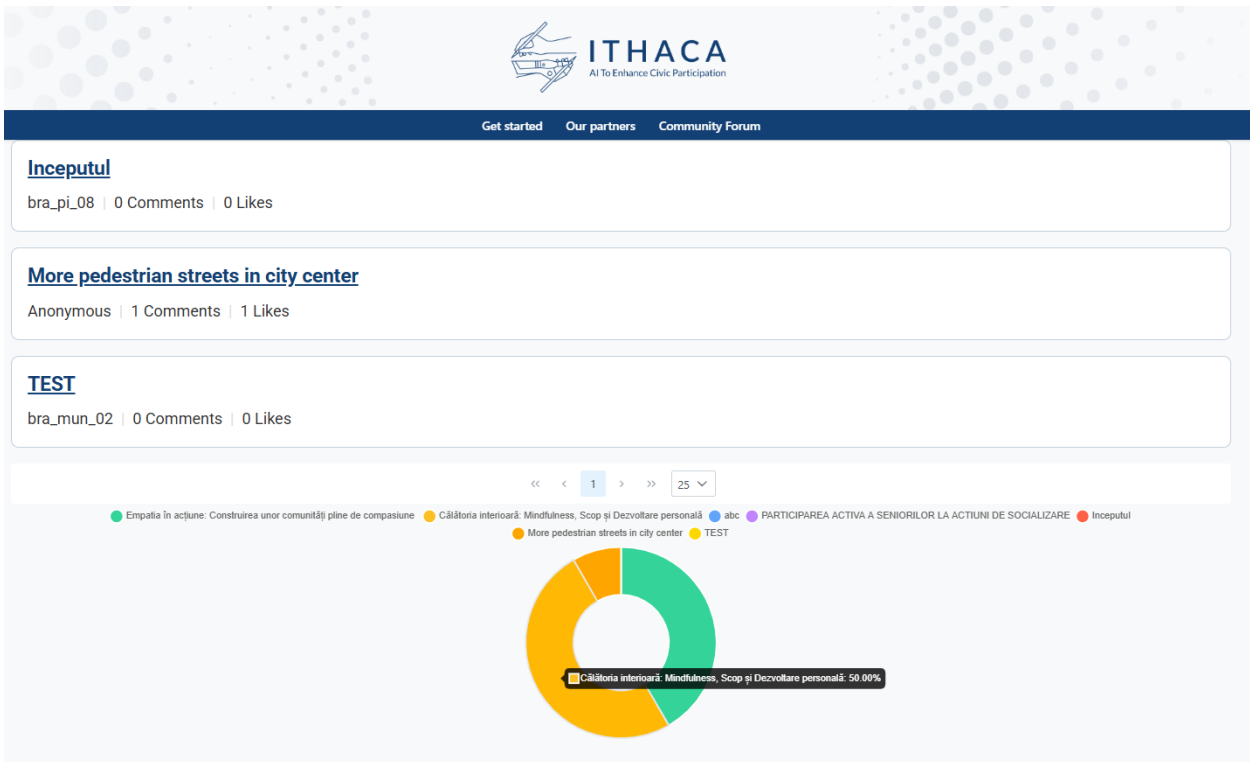
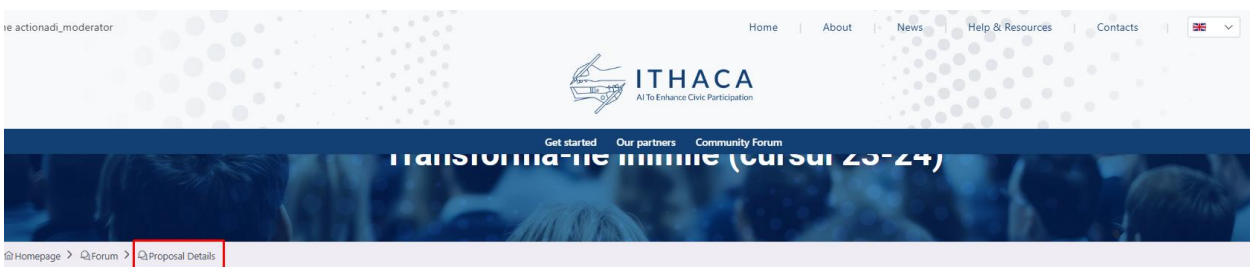


Figure 9 An example of the screen appearing in the ITHACA user interface. Upon selecting a public issue, a list of relevant proposals will appear.



[Transformă-ne inimile (cursul 23-24)]
Călătoria interioară: Mindfulness, Scop și Dezvoltare personală

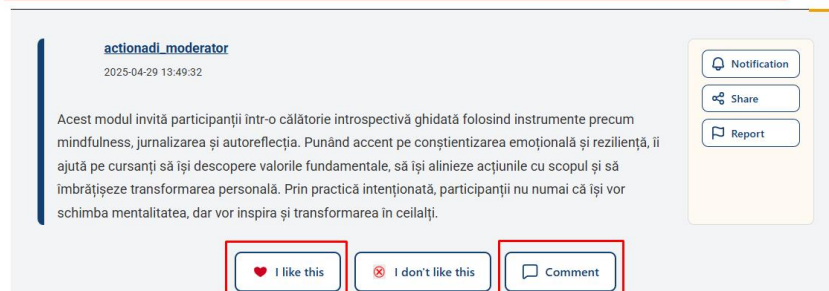


Figure 10 An example of the screen appearing in the ITHACA user interface when selecting a proposal for a public issue.

Upon selecting a proposal for a specific public issue, a screen such as the one in Figure 10, will appear. The user will be able to vote and/or post their argument to state their agreement or disagreement with a proposal.

4.1.8 T8 – AI Fairness

Provider: UPAT

Work Package: WP5, T5.1 – Fairness, Security and Privacy conformity assessment mechanisms

Short description of final module:

AI Fairness tool ensures conformity with the fairness principle, to promote equality, inclusivity, and oppose discrimination as well as render the evaluated AI system more trustworthy and unbiased.

The tool employs objective fairness metrics (e.g. treatment equality, disparate impact, between group generalized entropy etc.), to evaluate whether an AI model is being biased and unfair. An explainability mechanism was added to the AI Fairness tool to render its decision-making process transparent to the user.

The AI Fairness tool has already been integrated into the platform.

Relevant code snippets (refer to D5.4, Chapter 4 for extensive details):

evaluate_fairness.py

```
def evaluate_fairness(y_pred, val_labels_plus_vul):
    """
    Evaluate the fairness of a binary classification model.

    Args:
        y_pred (numpy.ndarray): Predicted labels of the model.
        val_labels_plus_vul (numpy.ndarray): Validation labels with vulnerability
    information.

    Returns:
        None

    Prints the fairness metrics based on the predicted labels and validation labels.

    Example:
        >>> y_pred = np.array([0, 1, 0, 1, 0])
        >>> val_labels_plus_vul = np.array([[0, 0], [1, 1], [0, 1], [1, 0], [0, 1]])

        >>> evaluate_fairness(y_pred, val_labels_plus_vul)
    """
    val_labels_df = pd.DataFrame(
```

```

        val_labels_plus_vul, columns=["toxicity", "vulnerability"]
    )
val_labels_bld = BinaryLabelDataset(
    df=val_labels_df,
    label_names=["toxicity"],
    protected_attribute_names=["vulnerability"],
    favorable_label=0,
    unfavorable_label=1,
)

predictions_plus_vul = np.concatenate(
    [y_pred.astype(int), val_labels_plus_vul[:, 1].reshape(-1, 1)], axis=1
)
predictions_plus_vul_df = pd.DataFrame(
    predictions_plus_vul, columns=["toxicity", "vulnerability"]
)
predictions_bld = BinaryLabelDataset(
    df=predictions_plus_vul_df,
    label_names=["toxicity"],
    protected_attribute_names=["vulnerability"],
    favorable_label=0,
    unfavorable_label=1,
)

metric = ClassificationMetric(
    val_labels_bld,
    predictions_bld,
    privileged_groups=[{"vulnerability": 0}],
    unprivileged_groups=[{"vulnerability": 1}],
)

print_fairness_metrics(metric, val_labels_df, predictions_plus_vul_df)

```

4.1.9 T9 - PPML tool

Provider: UPAT

Work Package: WP5, T5.1 – Fairness, Security and Privacy conformity assessment mechanisms

Short description of final module:

The Privacy Preserving Machine Learning (PPML) tool applies data concealment techniques to safeguard user privacy. More specifically, the tool utilizes a Differential Privacy method to perturb sensitive user data by adding random noise, so that it would not be possible for a potentially malicious actor to infer any personal, potentially identifying information in case of a potential attack via specifically engineered inputs to the model.

The tool is currently being integrated into the platform.

Relevant code snippets (refer to D5.4, Chapter 5 for extensive details):

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
import os
import shutil
from models import create_dp_model, PrivacyMetricsCallback
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow_privacy.privacy.analysis import (
    compute_dp_sgd_privacy_lib as compute_dp_sgd_privacy,
)
from tensorflow_privacy.privacy.privacy_tests.membership_inference_attack import (
    privacy_report,
)

from
tensorflow_privacy.privacy.privacy_tests.membership_inference_attack.data_structur
es
import (
    PrivacyMetric,
    AttackResultsCollection,
)
data_df = pd.read_csv("data/synthetic_fairness_dataset.csv")
data_df.head()
# Shuffle training dataframe
data_df = data_df.sample(
    frac=1, random_state=42
) # shuffle with random_state=42 for reproducibility

X = [re.sub("http://\S+|https://\S+", "", text) for text in data_df["tweet"]]
X = np.array(X)
vul = np.array(data_df["vulnerability"])
vul = vul.reshape(vul.shape[0], 1)
toxicity = np.array(data_df["Toxicity"])
toxicity = toxicity.reshape(toxicity.shape[0], 1)
y = np.concatenate([toxicity, vul], axis=1)

(
```

```

    train_sentences,
    val_sentences,
    train_labels_plus_vul,
    val_labels_plus_vul,
) = train_test_split(
    X, y, test_size=0.20, random_state=42
) # dedicate 20% of samples to validation set

train_labels = train_labels_plus_vul[:, 0]
train_labels = train_labels.reshape(train_labels.shape[0], 1)
val_labels = val_labels_plus_vul[:, 0]
al_labels = val_labels.reshape(val_labels.shape[0], 1)

train_sentences = np.array(train_sentences)
tf.get_logger().setLevel("ERROR")
tf.random.set_seed(42)

checkpoint_path = (
    "./models/differentially_private/model.{epoch:02d}-{val_accuracy:.4f}.ckpt"
)
checkpoint_dir = os.path.dirname(checkpoint_path)
if not os.path.exists(checkpoint_dir):
    os.makedirs(checkpoint_dir) # Create directory if it doesn't exist
else:
    shutil.rmtree(checkpoint_dir) # Remove directory to start new experiment
    os.makedirs(checkpoint_dir)

model_1 = create_dp_model(train_sentences)
# model_1.summary()

model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path,
    save_weights_only=True,
    save_freq="epoch",
    monitor="val_accuracy",
    mode="max",
    verbose=1,
    save_best_only=True,
)

EPOCHS = 200

```

```

BATCH_SIZE = 1024
epochs_per_report = 1
privacy_callback = PrivacyMetricsCallback(
    epochs_per_report,
    train_sentences,
    val_sentences,
    train_labels,
    val_labels,
    BATCH_SIZE,
    "Model 1",
)

lr_scheduler_callback = tf.keras.callbacks.ReduceLROnPlateau(factor=0.5,
    patience=15,
    verbose=1,cooldown=5, min_lr=1e-6)

early_stopping_callback = tf.keras.callbacks.EarlyStopping(
    monitor="val_accuracy", patience=10, start_from_epoch=20, verbose=1, mode="max"
)

all_reports = []
history_1 = model_1.fit(
    train_sentences,
    train_labels,
    epochs=EPOCHS,
    validation_data=(val_sentences, np.array(val_labels)),
    callbacks=[
        model_checkpoint_callback,
        privacy_callback,
        early_stopping_callback,
        lr_scheduler_callback,
    ],
    batch_size=BATCH_SIZE,
    shuffle=True,
)

all_reports.extend(privacy_callback.attack_results)

best = tf.train.latest_checkpoint(
    checkpoint_dir
) # Since save_best_only=True, latest model is also the best

```

```

model_1.load_weights(best)
model_1_evaluate = model_1.evaluate(val_sentences, val_labels)
print(
    compute_dp_sgd_privacy.compute_dp_sgd_privacy_statement(
        number_of_examples=train_sentences.shape[0],
        num_epochs=int(best.split('-')[0].split('.')[0]),
        batch_size=BATCH_SIZE,
        noise_multiplier=1.0,
        delta=1e-5,
        used_microbatching=False,
    )
)

results = AttackResultsCollection(all_reports)

privacy_metrics = (
    PrivacyMetric.AUC,
    PrivacyMetric.ATTACKER_ADVANTAGE,
    PrivacyMetric.PPV,
) # AUC, ATTACKER_ADVANTAGE, PPV, EPSILON_LOWER_BOUND

epoch_plot = privacy_report.plot_by_epochs(results,
privacy_metrics=privacy_metrics)

if not os.path.exists('./results/dp_model/'):
    os.makedirs('./results/dp_model/') # Create directory if it doesn't exist
else:
    shutil.rmtree('./results/dp_model/') # Remove directory to start new experiment
    os.makedirs('./results/dp_model/')
results.save('./results/dp_model/')
plt.show()

def plot_graphs(history, string):
    plt.plot(history.history[string])
    plt.plot(history.history["val_" + string])
    plt.xlabel("Epochs")
    plt.ylabel(string)
    plt.legend([string, "val_" + string])
    plt.show()

plot_graphs(history_1, "accuracy")
plot_graphs(history_1, "loss")

```

4.1.10 T10 – AI Cybersecurity tool

Provider: UPAT

Work Package: WP5, T5.1 – Fairness, Security and Privacy conformity assessment mechanisms

Short description of final module:

An AI Cybersecurity tool to detect possible security breaches and threats within the AI models. An open-source tool named ModelScan (<https://modelscan.ai>) that scans for malicious code that may impose security vulnerabilities in AI systems is proposed as the AI Cybersecurity tool.

Relevant code snippets (refer to D5.4, Chapter 6 for extensive details):

```
# Example of creating and scanning a model

class SafeModel(tf.keras.Model):
    def __init__(self):
        super(SafeModel, self).__init__()
        self.conv1 = Conv2D(32, (3, 3), input_shape=(28, 28, 1))
        self.pool1 = MaxPooling2D((2, 2))
        self.conv2 = Conv2D(64, (3, 3))
        self.pool2 = MaxPooling2D((2, 2))
        self.flatten = Flatten()
        self.d1 = Dense(128)
        self.d2 = Dense(10, activation='softmax')

    def call(self, x):
        x = self.conv1(x)
        x = self.pool1(x)
        x = self.conv2(x)
        x = self.pool2(x)
        x = self.flatten(x)
        x = self.d1(x)
        return self.d2(x)

safe = SafeModel()
safe.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
safe.fit(train_images, train_labels, epochs=1)
safe.save('safemodel')
```

Upon saving the model, we run ModelScan against it. ModelScan performs a thorough analysis by scanning various components of the saved TensorFlow model.

```
modelscan -p ./safemodel
```

4.1.11 T11 – Simplification Tool

Provider: SIMAVI

Work Package: WP3

Short description of final module:

This API simplifies complex texts by leveraging a multilingual transformer model (**mbart-large-50**). It detects the text's language (or uses a specified language), processes the input to reduce complexity, and generates a simplified version of the text while maintaining its core meaning. The API supports multiple languages, including English, Romanian, and Slovak, and is optimized for accurate and concise outputs with features like beam search and repetition control.

Installation and configuration:

Follow the steps below to set up and run the text simplification service based on the provided script:

1. Prerequisites

Ensure you have the following installed on your system:

- **Python 3.8+**
- **pip** (Python package manager)
- **virtualenv** (for creating isolated environments)

2. Installation Steps

1. **Clone or Download the Script:** Save the script above as `summarize.py` in a directory of your choice.

```
DetectorFactory.seed = 0

# Initialize FastAPI app
app = FastAPI()

# Load the multilingual model and tokenizer
model_name = "facebook/mbart-large-50" # Multilingual model
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

# Initialize the pipeline
simplifier = pipeline(
    "text2text-generation",
    model=model_name,
    tokenizer=model_name,
)

# Language codes for mbart-large-50
LANGUAGE_CODES = {
```

```

    "ro": "ro_RO", # Romanian
    "sk": "sk_SK", # Slovak
    "en": "en_XX", # English (default)
}

# Request and Response Models
class SimplificationRequest(BaseModel):
    content: str # Text to simplify
    language: str = None # Optional: language override

class SimplificationResponse(BaseModel):
    simplified_content: str # Simplified output

@app.post("/simplify", response_model=SimplificationResponse)
def simplify_text(request: SimplificationRequest):
    try:
        # Detect the language if not provided
        if not request.language:
            detected_lang = detect(request.content)
            lang_code = detected_lang
        else:
            lang_code = request.language.lower()

        # Map the detected/provided language to mbart-large-50 language codes
        tgt_lang = LANGUAGE_CODES.get(lang_code, "en_XX")

        # Set the forced_bos_token_id for the target language
        model.config.forced_bos_token_id = tokenizer.lang_code_to_id[tgt_lang]

        # Compute dynamic max_length
        input_length = len(request.content.split()) # Approximate word count
        max_length = min(int(input_length * 1.2), 1024) # Max length capped at 1024
        min_length = max(int(input_length * 0.8), 10) # Min length at least 10
words

        # Simplify the input content
        simplified = simplifier(
            request.content,
            max_length=max_length,
            min_length=min_length,
            do_sample=False,
            num_beams=5, # Use beam search
            no_repeat_ngram_size=3, # Prevent repetition
        )

```

```
# Extract the simplified text
simplified_text = simplified[0]["generated_text"]

return SimplificationResponse(simplified_content=simplified_text)
except Exception as e:
    raise HTTPException(status_code=500, detail=f"Error during simplification:
{str(e)}")
```

3. Create a Virtual Environment:

```
bash
```

```
python3 -m venv venv
```

4. Activate the Virtual Environment:

```
bash
```

```
source venv/bin/activate
```

5. Install Dependencies: Install the required Python libraries:

```
bash
```

```
pip install fastapi uvicorn transformers langdetect torch gunicorn
```

Running (on Ubuntu):

From within the same directory as the above `simplify.py` file

```
gunicorn -w 4 -k uvicorn.workers.UvicornWorker --bind 10.222.30.234:8000
simplify:app &
```

Produced messages (topics):

N/A

Consumed Messages (topics):

N/A

How to test the system:

The api is only accessible from behind the vpn:

POST <http://10.222.30.234:8001/simplify>

Body:

```
{  
  "content": "A very complex text that needs to be summarized"  
}
```

4.1.12 T12 – Summarization Tool

Provider: SIMAVI

Work Package: WP3

Short description of final module:

This API summarizes complex texts by leveraging a multilingual transformer model (**mbart-large-50**). It detects the text's language (or uses a specified language), processes the input to generate a concise and coherent summary, and supports multiple languages, including English, Romanian, and Slovak. The API is designed for precision and fluency, ensuring the summarized content retains the core meaning of the original text.

Installation and configuration:

Follow the steps below to set up and run the text summarization service based on the provided script:

1. Prerequisites

Ensure you have the following installed on your system:

- **Python 3.8+**
- **pip** (Python package manager)
- **virtualenv** (for creating isolated environments)

2. Installation Steps

1. Clone or Download the Script:

Save the script as `summarize.py` in a directory of your choice.

```
from fastapi import FastAPI, HTTPException  
from pydantic import BaseModel  
from transformers import pipeline, AutoTokenizer, AutoModelForSeq2SeqLM  
from langdetect import detect, DetectorFactory  
  
# Ensure language detection is consistent  
DetectorFactory.seed = 0  
  
# Initialize FastAPI app  
app = FastAPI()  
  
# Load the mBART model and tokenizer
```

```

model_name = "facebook/mbart-large-50"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

# Translation models
translation_models = {
    "ro": {
        "to_en": "Helsinki-NLP/opus-mt-roa-en",
        "from_en": "Helsinki-NLP/opus-mt-en-ro",
    },
    "sk": {
        "to_en": "Helsinki-NLP/opus-mt-sk-en",
        "from_en": "Helsinki-NLP/opus-mt-en-sk",
    },
    "default": {
        "to_en": "Helsinki-NLP/opus-mt-ROMANCE-en",
        "from_en": "Helsinki-NLP/opus-mt-en-ROMANCE",
    },
}

# Request and Response Models
class SimplificationRequest(BaseModel):
    content: str # Text to simplify
    language: str = None # Optional language code (if not provided, detect it)

class SimplificationResponse(BaseModel):
    simplified_content: str # Simplified output

def split_into_chunks(text, max_tokens=512):
    """
    Splits text into smaller chunks that do not exceed the model's max token limit.
    """
    words = text.split()
    for i in range(0, len(words), max_tokens):
        yield " ".join(words[i:i + max_tokens])

@app.post("/simplify", response_model=SimplificationResponse)
def simplify_text(request: SimplificationRequest):
    try:
        # Detect the language if not provided
        if not request.language:

```

```

        detected_lang = detect(request.content)
        lang_code = detected_lang
    else:
        lang_code = request.language.lower()

    # Load language-specific translation models
    translation_model = translation_models.get(
        lang_code, translation_models["default"]
    )
    translator = pipeline("translation", model=translation_model["to_en"])
    back_translator = pipeline("translation",
model=translation_model["from_en"])

    # Map the detected/provided language to mbart-large-50 language codes
    tgt_lang = f"{lang_code}_{lang_code.upper()}"
    if tgt_lang not in tokenizer.lang_code_to_id:
        tgt_lang = "en_XX" # Default to English if the language is unsupported

    # Set the tokenizer to use the target language
    model.config.forced_bos_token_id = tokenizer.lang_code_to_id[tgt_lang]

    # Split input text into manageable chunks
    chunks = list(split_into_chunks(request.content, max_tokens=512))
    simplified_chunks = []

    for chunk in chunks:
        # Step 1: Translate the chunk to English
        translated_chunk = translator(chunk, max_length=512,
truncation=True)[0]["translation_text"]

        # Step 2: Simplify the translated chunk
        simplifier = pipeline("text2text-generation", model=model,
tokenizer=tokenizer)
        simplified = simplifier(
            translated_chunk,
            max_length=150,
            min_length=20,
            do_sample=False,
            num_beams=5,
            no_repeat_ngram_size=3,
        )
        simplified_text = simplified[0]["generated_text"]

```

```

        # Step 3: Translate the simplified text back to the original language
        final_simplified = back_translator(simplified_text, max_length=512,
truncation=True)[0]["translation_text"]

        simplified_chunks.append(final_simplified)

    # Combine all simplified chunks into a single output
    full_simplified_text = " ".join(simplified_chunks)

    return SimplificationResponse(simplified_content=full_simplified_text)
except Exception as e:
    raise HTTPException(status_code=500, detail=f"Error during simplification:
{str(e)}")

```

2. Create a Virtual Environment:

```
bash
```

```
python3 -m venv venv
```

3. Activate the Virtual Environment:

```
bash
```

```
source venv/bin/activate
```

4. Install Dependencies:

Install the required Python libraries:

```
bash
```

```
pip install fastapi uvicorn transformers langdetect torch gunicorn
```

Running (on Ubuntu):

Start the service using Gunicorn:

```
bash
```

```
gunicorn -w 4 -k uvicorn.workers.UvicornWorker --bind 10.222.30.234:8000
summarize:app &
```

The API will now be accessible on port 8000 behind the VPN.

Produced messages (topics):

N/A

Consumed Messages (topics):

N/A

How to test the system:

The API is accessible only from behind the VPN. Test it using the following steps:

Request

```
POST http://10.222.30.234:8000/summarize
```

Headers:

```
json
```

```
{  
  "Content-Type": "application/json"  
}
```

Body:

```
json
```

Copy code

```
{  
  "content": "A very long and complex text that needs to be summarized into something  
concise.",  
  "language": "en"  
}
```

Response

```
json
```

```
{  
  "summarized_content": "A concise summary of the provided complex text."  
}
```

4.1.13 T13 – Translation Tool

Provider: KT

Work Package: WP3 (to be used as a complementary tool)

Short description of final module:

A tool that will translate a text to the selected language. Translation capabilities are from Romanian and Slovak to English and from English to Romanian and Slovak.

Relevant code snippets:

```
def translate_text(text, tokenizer, model):
    inputs = tokenizer(text, return_tensors="pt", padding=True)
    translated = model.generate(**inputs)
    return tokenizer.decode(translated[0], skip_special_tokens=True)

class TranslateTextView(views.APIView):
    def post(self, request, format=None):
        # Models to use assuming that target language is English:
        models_from = {
            'ro': "Helsinki-NLP/opus-mt-ROMANCE-en", #"Helsinki-NLP/opus-mt-ro-en"
            'sk': "Helsinki-NLP/opus-mt-sk-en"
        }
        # Models to use assuming that language of origin is English:
        models_to = {
            'ro': "Helsinki-NLP/opus-mt-en-ro",
            'sk': "Helsinki-NLP/opus-mt-en-sk"
        }

        if request.data['from'] == 'en':
            model_name = models_to[request.data['to']]
        else:
            model_name = models_from[request.data['from']]

        tokenizer = MarianTokenizer.from_pretrained(model_name,
            token=settings.HUGGING_TOKEN)
        model = MarianMTModel.from_pretrained(model_name,
            token=settings.HUGGING_TOKEN)

        translated_text = translate_text(request.data['text'], tokenizer, model)
        return Response({'translation':translated_text}, status=status.HTTP_200_OK)
```

4.1.14 T14 – Chatbot

Provider: KT

Work Package: WP3 (to be used as a complementary tool)

Short description of final module:

A tool to assist platform navigation and provide information to users, enhancing user experience.

The chatbot model is self-hosted, which means that all information are not shared with third party AI APIs.

Relevant code snippets:

```
def generate(prompt):
    model_name = "google/gemma-3-1b-it"

    tokenizer = AutoTokenizer.from_pretrained(model_name,
        use_auth_token=settings.HUGGING_TOKEN)

    model = Gemma3ForCausalLM.from_pretrained(model_name,
        use_auth_token=settings.HUGGING_TOKEN).cpu()

    input_ids = tokenizer.encode(prompt + tokenizer.eos_token, return_tensors='pt')
    attention_mask = torch.ones_like(input_ids)

    output_ids = model.generate(input_ids, max_length=1000,
        pad_token_id=tokenizer.eos_token_id, attention_mask=attention_mask)

    response = tokenizer.decode(output_ids[0], skip_special_tokens=True)

    return response

class ChatbotView(views.APIView):
    def post(self, request, format=None):
        answer = generate(request.data['prompt'])

        ChatbotPrompt.objects.create(prompt=request.data['prompt'], answer=answer)

        return Response({'answer': answer}, status=status.HTTP_200_OK)
```

5. Demonstrator Operation and Key Functionalities

5.1 Overview

In this chapter, we focus on the system's visible layer, namely the user interface elements. The backend, which is not directly observed, is the system's most critical component. Its presence is only evident through the results displayed in the UI.

5.2 Main actions

The action flow shown in the upcoming screenshots is available on the platform and is fully described in the following subchapters.

5.1.1 Login/register

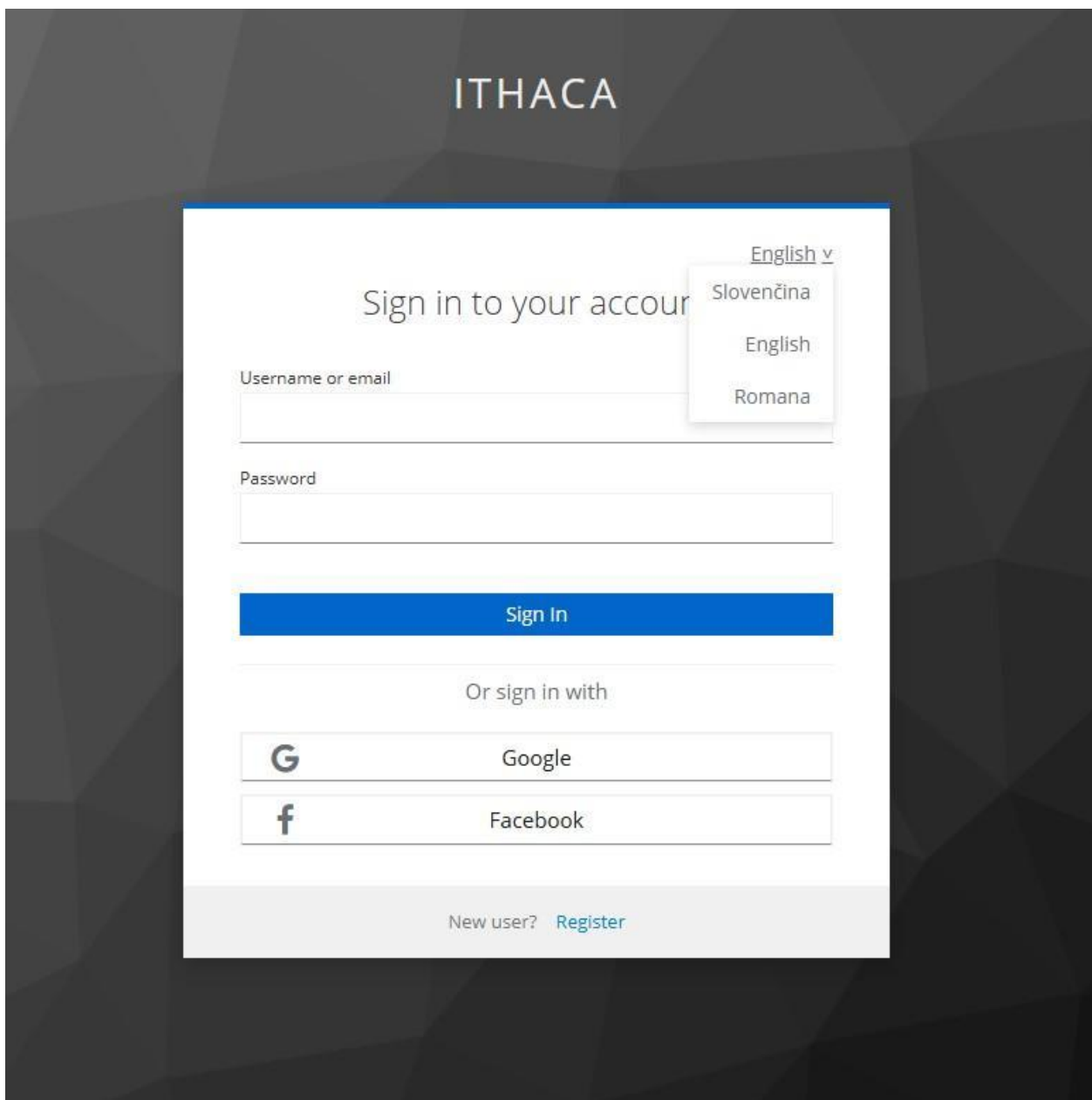


Figure 11 Login / Register

The ITHACA login page appears when a user accesses the application from a browser. At this stage, the user is redirected to Keycloak for authentication.

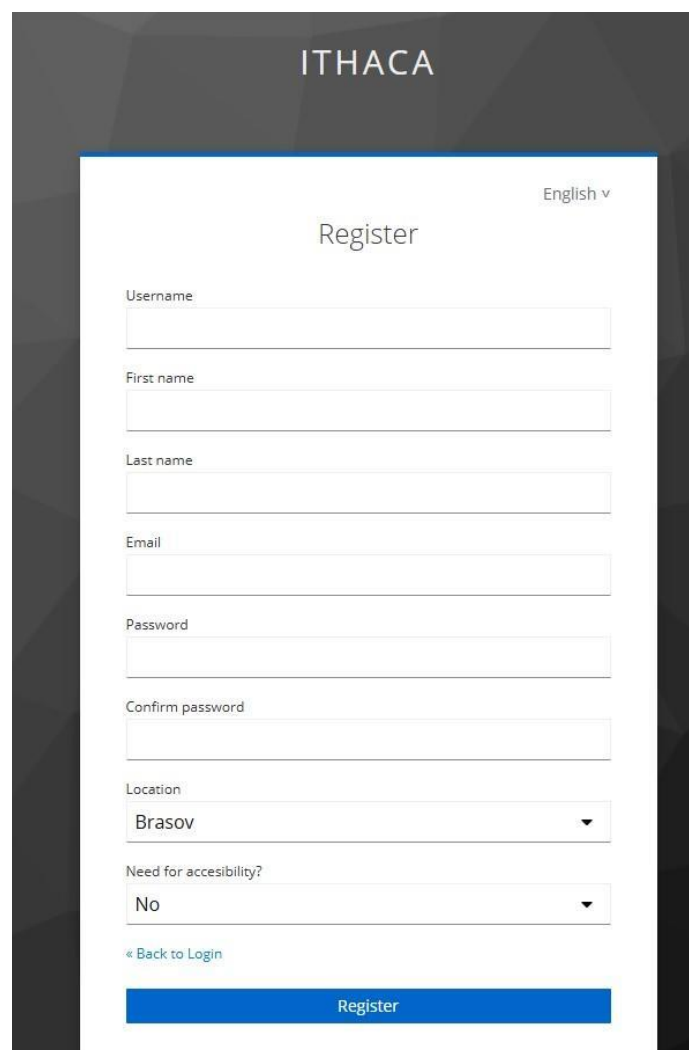
From the user's perspective, the flow is as follows:

1. The user visits the application in a browser and attempts to access a protected resource.
2. The application redirects the user to the Keycloak login page.
3. The user provides their username and password.
4. Keycloak authenticates the credentials.
5. Upon successful authentication, Keycloak redirects the user back to the protected resource within the application.

Keycloak supports different languages for the menus, as seen in the screenshot, like Slovak, English, Romanian.

Following login, every page is rendered within a template that includes a menu on the left (in either an extended or short form). On the right, the relevant information for each selected command is displayed.

Keycloak also provides a register form where a new user can be created. User can select Brasov or Martin as location and all the content will be displayed based on the selection from Brasov or Martin.



ITHACA

English v

Register

Username

First name

Last name

Email

Password

Confirm password

Location
Brasov ▼

Need for accessibility?
No ▼

[« Back to Login](#)

Register

Figure 12 Registration screen

5.1.2 The main dashboard

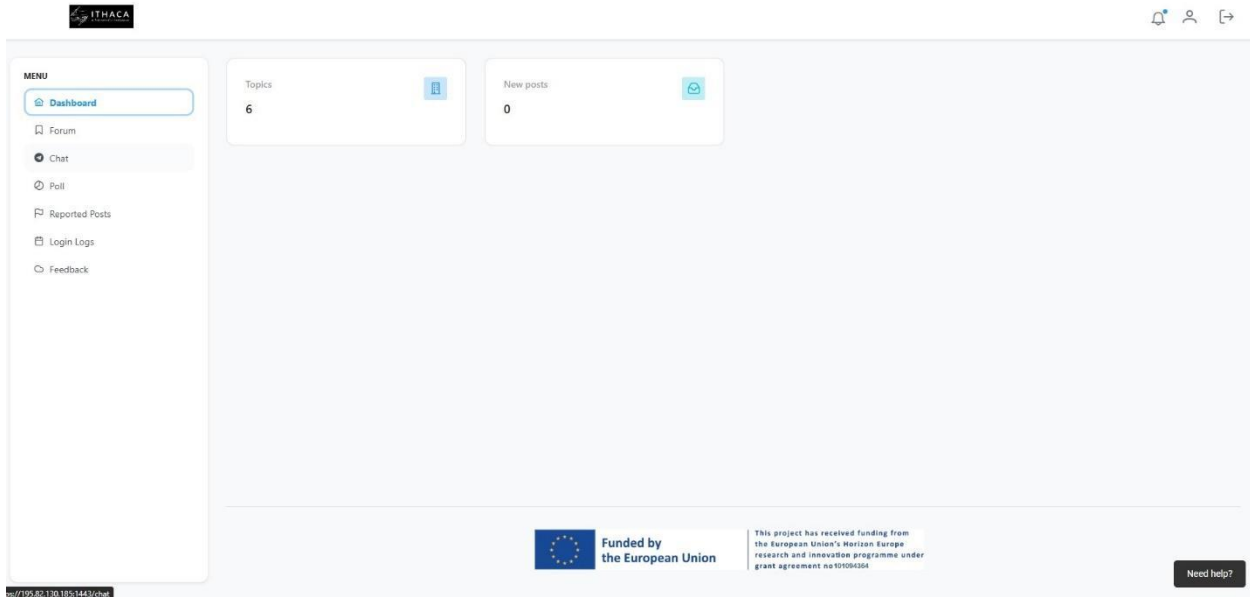


Figure 13 Main dashboard

The main dashboard is the place where user gets redirected after a successful login. It presents different statistics like number of topics, most view topic in the last week, most discussed issues of the last week and different statistics.

5.1.3 Forum page

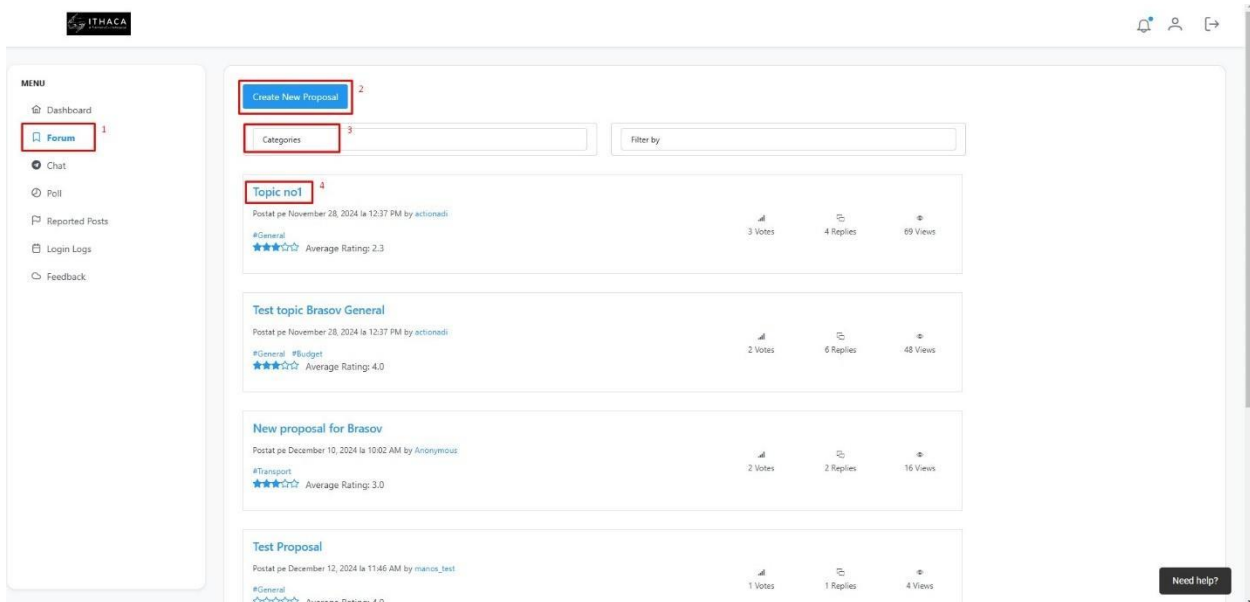


Figure 14 Forum page

Next, we have the forum page (1), where users can see all the proposals that are created by users or automatically created following the crawling of the content from the Brasov or Martin area.

The proposals can be filtered (2) by categories like general, transportation, budget, etc.

Create New Proposal ✕

Title:

Content:

No file chosen

Categories:
 ▾

Post Anonymously

Figure 15 Creation of a new proposal

Users can create new proposals (2) by entering the title, the content that will be the first post of the topic, optional can add pictures to it, select categories from General, Budget, Transportation, and it can post anonymously.

When “create proposal” is clicked, the text is checked by the toxicity tool, the pictures are checked by another tool that verifies if the content is safe then if successful the new proposal is created.

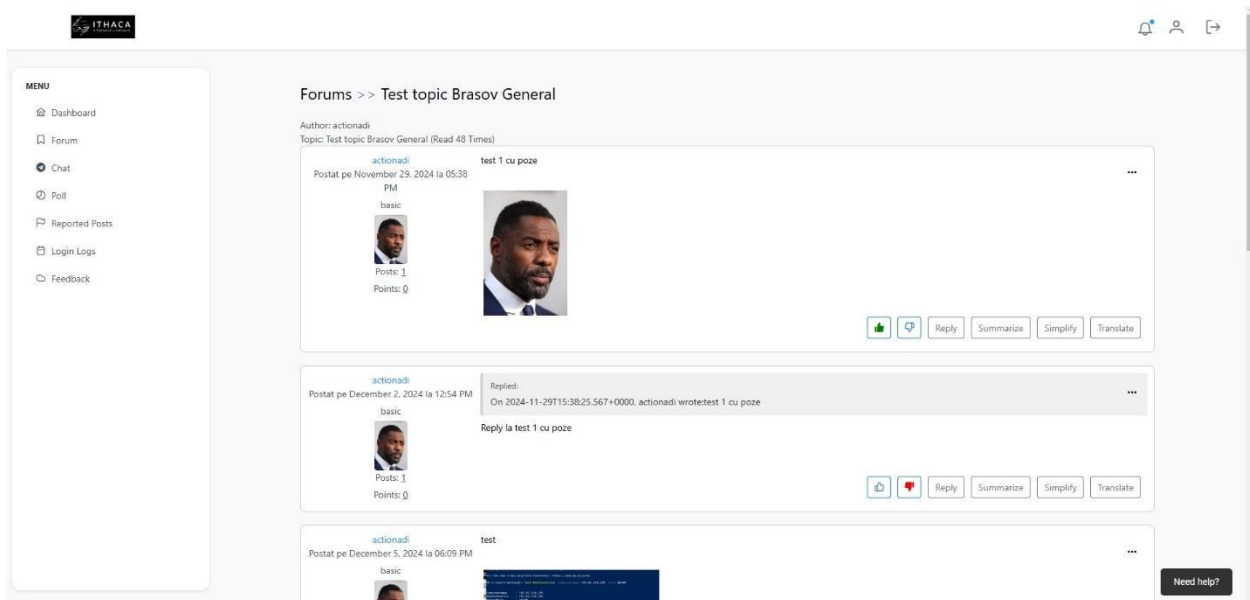


Figure 16 Toxicity filter

By clicking the forum title, a new page is opened with the posts of that topic. Info of the person of a new post is displayed (unless selected anonymously) like avatar, posts, points. The content of the post can contain text, pictures, and replies to other users' posts.

In the top right of the post there is a button with an icon of 3 dots where the user can report a post that is considered to have inappropriate content.

In the bottom right of each post there are several buttons, like/dislike buttons, Reply, Summarize (a dialog will be opened with the initial text and a summarization of the complex text), Simplify (a dialog will be opened with the initial text and a simplification of the complex text), Translate (a dialog will be opened with the initial text and the translation in the language that the user has selected for the app).

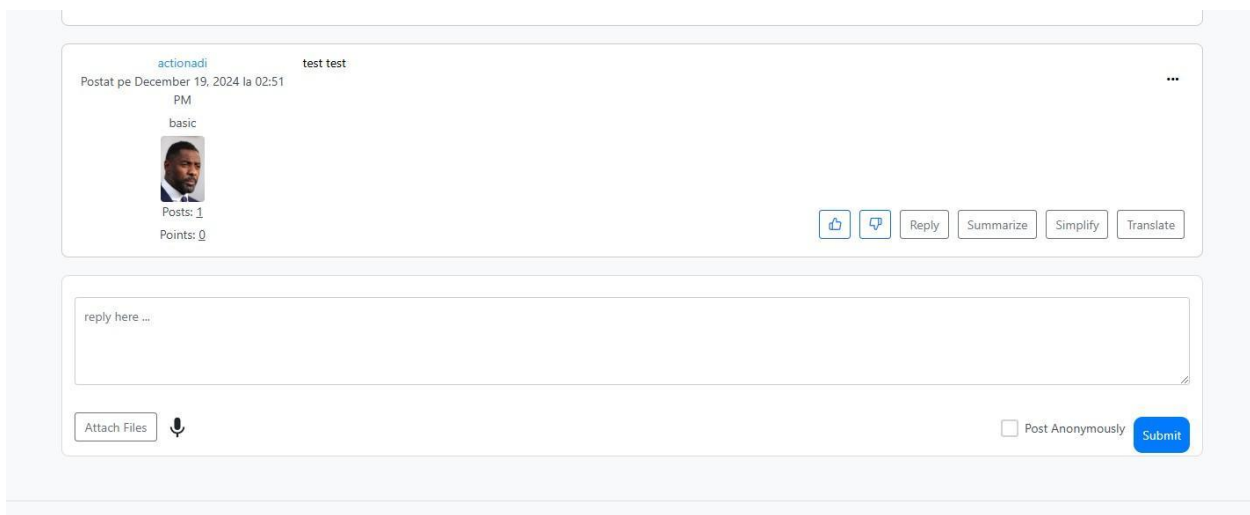


Figure 17 Reply, Summarize, Simplify, Translate buttons

At the bottom of the posts there is a form where users can post replies, attach files, use speech to text microphone button that transforms the speech into text and by clicking "Post Anonymously" user can hide his identity.

5.1.4 Chat page

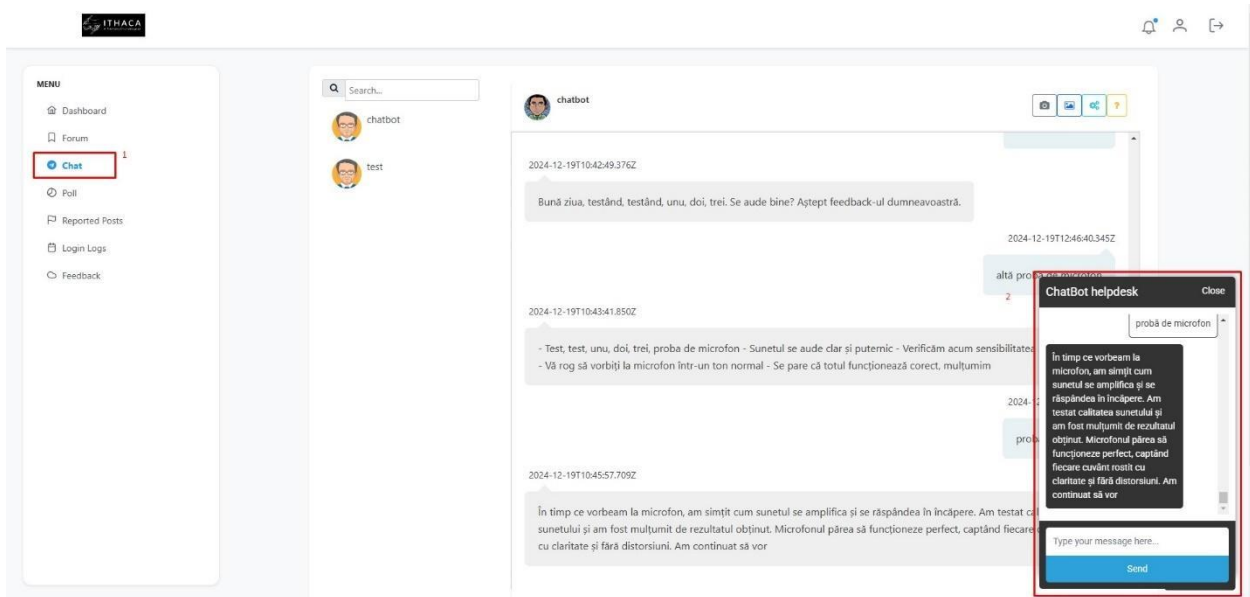


Figure 18 Chat Page

Next, there is a chat page, where the user can chat with a chatbot and receive information on different topics that the chatbot is trained for.

The chat with the chatbot can be opened in part of the application, by clicking the need help button on the bottom right.

5.1.5 Reported posts

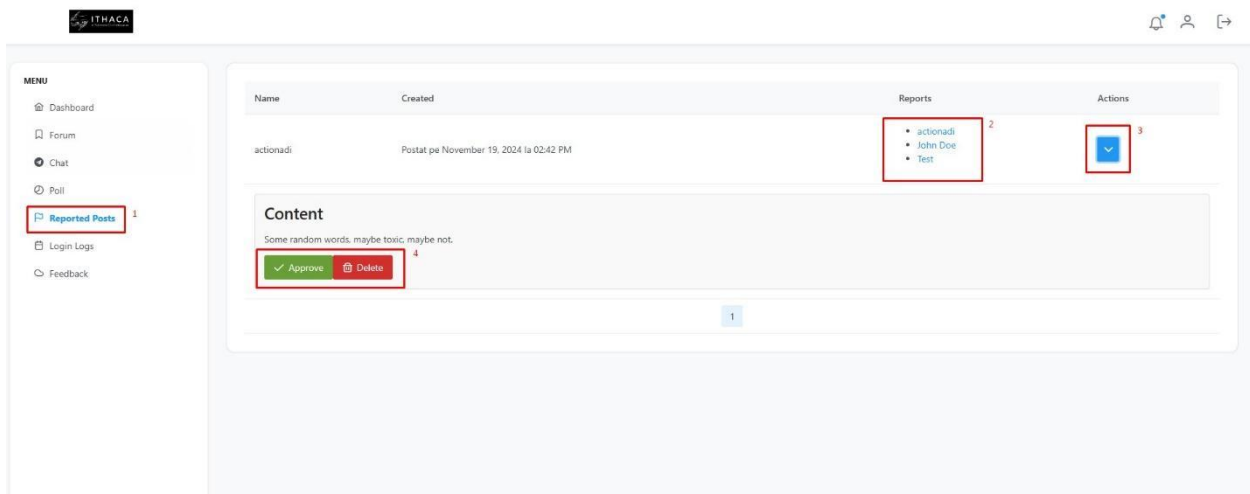


Figure 19 Reported posts

Further, we have the reported posts page, where a table with all the posts that are marked for review. A post becomes flagged when it gets reported 3 times. By clicking the actions button (3) the content of the post is displayed along with the approve and delete button. The approve button clears the reports and marks the post as safe again. Delete button deletes the post.

This page will be available to users of administrative level.

5.1.6 Login Logs page

Username	IP Address	Created Date
acionadi	195.82.130.6	12/19/24, 12:39 PM
acionadi	86.123.207.13	12/19/24, 4:04 PM
manos	62.1.92.104	12/31/24, 12:52 AM
acionadi	86.123.207.13	1/14/25, 2:30 PM
acionadi	86.123.207.13	1/14/25, 2:46 PM

Total Logs: 25

Showing 21 to 25 of 25

Figure 20 Login logs page

Next, we have the login logs page, where all the login details are stored, the username, ip and time of login. It can be useful to detect multiple accounts from the same user.

This page will be available to users of administrative level.

5.1.7 Feedback page

Name	Created	Tags	Title	Content
acionadi	Postat pe December 19, 2024 la 03:52 PM	#General #Appearance	Feedback Title	Feedback content

1

Figure 21 Feedback page

Next, we have a feedback page, where users can post feedback on the app, with categories like general, appearance, accessibility.

5.1.8 Profile page

The screenshot shows the ITHACA user profile page. On the left is a 'MENU' sidebar with links to Dashboard, Forum, Chat, Poll, Reported Posts, Login Logs, and Feedback. The main area contains a profile form with the following fields: Last Name (text input with 'D'), Accessibility (dropdown menu with 'Yes'), Minority (dropdown menu with 'Yes'), and Language (dropdown menu with 'English'). Below the form is a profile picture of a man in a suit, with a 'Choose File' button and 'No file chosen' text. A blue 'Update profile' button is at the bottom. A 'Need help?' button is in the bottom right corner. A red box highlights the user profile icon in the top right corner.

Figure 22 Profile page

On the top right of the menu there is a button where the user profile can be accessed. Can set the name, first name, Accessibility (if selected yes special accessibility functions are activated), language selection between English, Romanian, Slovak, update the avatar are possible.

6. Challenges and Limitations

6.1 Technical Challenges

1. Integration of Heterogeneous Technologies:

- o Bringing together various technologies, each contributed by different partners, posed significant integration challenges.
- o Ensuring these components communicate effectively required defining clear APIs, protocols, and data exchange mechanisms.

2. Interoperability Across Modules:

- o The platform incorporates tools like evidence extraction, chatbots, gamification mechanisms, and argument visualization, each developed by different teams. Achieving seamless interoperability between these modules required rigorous interface design and testing.

3. Data Privacy and Security:

- o Handling sensitive data necessitated adherence to strict security standards, especially while implementing multi-factor authentication through **FortiClient VPN** and **Keycloak**.

4. Scalability and Performance Optimization:

- o Accommodating large datasets and high user loads demanded scalable architecture, including efficient use of **Docker** for microservices and orchestration.

5. **Localization and Multilingual Support:**

- o Supporting multiple languages (e.g., English, Slovak, Romanian) across the UI and backend systems required careful handling of text encoding, translations, and cultural nuances.

7. **Conclusions and Next Steps**

Conclusions:

The ITHACA platform's operational prototype represents a significant milestone in the project's ambition to harness Artificial Intelligence to enhance civic participation. The integration of numerous advanced modules - ranging from evidence extraction and topic clustering to gamification, AI fairness, and multilingual services - has resulted in a powerful, modular, and extensible digital ecosystem.

The project has successfully delivered a technically mature and user-oriented platform that aligns with the European Interoperability Framework and responds to real-world needs identified in pilot cities. Notably, the platform supports multilingual interaction (English, Romanian, Slovak), modular tool deployment, and advanced AI features such as toxicity detection, summarization, and privacy-preserving analytics - all implemented in a secure, cloud-hosted environment.

Despite the heterogeneity of technologies and contributors, the consortium demonstrated strong coordination and problem-solving skills, overcoming significant integration and deployment challenges. The adoption of containerization, microservices, and CI/CD pipelines has laid a robust foundation for scalable, maintainable, and secure deployment.

Crucially, the project has also shown sensitivity to ethical and societal concerns through its fairness, transparency, and privacy-preserving mechanisms. These considerations, embedded in the technical design, affirm the platform's commitment to trustworthy and human-centric AI.

Next Steps:

1. **System Optimization and Scalability:**

- o Refine backend performance to better accommodate real-time interaction and increased traffic in future large-scale pilot deployments.
- o Improve data processing pipelines to support more languages and larger datasets efficiently.

2. **Enhanced User-Centered Design:**

- o Integrate detailed feedback from pilot users to fine-tune usability and accessibility.
- o Expand personalization features and optimize user journeys within the platform (e.g., adaptive content, intelligent navigation support).

3. **Finalization of Remaining Components:**

- o Complete the implementation of pending tools such as the backend of the gamification engine and the Public AI Register.
- o Deepen integration between modules to improve user experience continuity and reduce cognitive load.

4. **Extended Testing and Evaluation:**

- o Conduct structured usability tests, stress tests, and security audits across diverse user groups.

- Align validation processes with WP6 and WP7 efforts for impact assessment and iterative development.

5. Deployment and Stakeholder Engagement:

- Begin structured pilot rollouts in Brasov and Martin, engaging both municipal authorities and citizens.
- Establish feedback loops with stakeholders to continuously refine features and ensure adoption.

6. Strategic Roadmapping and Sustainability:

- Develop a long-term roadmap for technological upgrades, feature evolution, and exploitation pathways.
- Explore potential for policy uptake and integration with other European civic tech initiatives.

8. References

1. Keycloak: An open-source identity and access management solution for modern applications and services.

[Keycloak Official Documentation](#)

2. Docker: A platform for developing, shipping, and running applications in containers.

[Docker Official Documentation](#)

3. FortiClient VPN: A robust VPN solution for secure access.

[FortiClient Documentation](#)

4. MongoDB: A flexible, document-oriented NoSQL database.

[MongoDB Official Documentation](#)