Ben-Gurion University of the Negev

Faculty of Engineering Sciences

Department of Software and Information Systems Engineering

Vox Populi Platform

קול האנשים

Final Year Seminar Project

Submitted by:

Ron Sharabi

Roey Fabian

Amit Cohen

Eran Fishbein

Supervised by:

Dr. Nir Grinberg

June 2025

Abstract

Abstract:

The evolution of surveys has made them an indispensable tool in multiple fields, including marketing, public opinion research, healthcare, and education. Surveys have evolved from door-to-door interviews to digital platforms, significantly expanding research reach across diverse populations. The rise of smartphones and messaging applications created unprecedented opportunities for real-time data collection, allowing researchers to conduct researches in multiple fields and subjects with richer data. However, with the emergence of Large Language Models, social networks have increasingly restricted access to private data, limiting researchers' ability to study authentic communication patterns. To address this challenge, we developed the Vox Populi Platform, leveraging the Matrix protocol to establish connections with participants' messaging accounts. The platform uses Matrix bridges to connect with WhatsApp, Telegram, and Signal through a web application. Users maintain full control over which conversations they share for research purposes. Selected communications are processed through an anonymization pipeline using Python-based tools specialized for Hebrew Named Entity Recognition. The platform's unique advantages include seamless integration of multiple messaging protocols, granular user control over data sharing, and Hebrew language processing. This enables researchers to access authentic, diverse, and ethically-sourced communication data while respecting participant privacy.

תקציר:

ההתפתחות של סקרים הפכה אותם לכלי הכרחי בתחומים רבים כולל שיווק, סקרי דעת קהל, בריאות וחינוך. איסוף נתונים על יד סקרים התפתח עם השנים מאיסוף באופן אישי לפלטפורמות דיגיטליות שהרחיבו משמעותית את המחקר על פני אוכלוסיות מגוונות. הופעתם של הסמארטפונים ואפליקציות העברת המסרים שיפרה את יכולת איסוף הנתונים אף יותר ואפשרה לחוקרים לבצע מחקרים בתחומים רחבים יותר ובשימוש בנתונים עשירים יותר. עם הופעת מודלי השפה הגדולים, רשתות חברתיות הגבילו יותר ויותר את הגישה לנתונים פרטיים, דבר שהגביל את יכולתם של חוקרים לבצע את עבודתם. כדי להתמודד עם האתגר הזה, פיתחנו את הפלטפורמה הזו תוך שימוש בפרוטוקול מטריקס ליצירת קשר עם חשבונות שונים של משתתפים. הפלטפורמה משתמשת בגשרים של מטריקס כדי להתחבר לוואטסאפ, טלגרם וסיגנל דרך עמוד אינטרנט, כאשר למשתמשים יש שליטה מלאה באילו שיחות הם משתפים למטרות מחקר. השיחות הנבחרות עוברות אנונמיזציה מקיפה באמצעות כלים שמיושמים בשפת פייתון ובמודל שפה שמתמחה בזיהוי ישויות בשפה העברית. היתרונות היחודיים של פלטפורמה זו כוללים אינטגרציה חלקה של מספר יישומים להעברת מסרים בעלי פרוטוקולי תקשורת שונים, בקרת משתמש אישית על שיתוף הנתונים, עיבוד ואנונימיזציה בשפה העברית.

Table of Contents

1 Introduction

1.1 Background

Data collection methodologies have undergone fundamental transformations over the past century, revolutionizing how researchers gather insights across multiple disciplines. Traditional research approaches, once limited by geographical boundaries and resource constraints, have expanded into digital ecosystems that enable global reach and real-time data acquisition.

Modern survey infrastructure encompasses online questionnaires, email-based studies, and social media polling mechanisms, dramatically expanding both the scale and speed of data collection while significantly reducing associated costs.

However, this technological advancement has introduced new complexities alongside its benefits. While digital platforms allow researchers to gather data from substantially larger and more diverse populations in considerably less time than traditional methods required, they have simultaneously increased risks associated with data authenticity and reliability.

Universal access to global news streams and the ubiquity of opinion-sharing platforms have created an environment where individuals routinely engage with real-time information and participate in worldwide discussions through their mobile devices. Smartphones, social media networks, and instant messaging applications have transformed ordinary citizens into potential data contributors, generating vast streams of authentic conversational data that represents unprecedented opportunities for more current, accurate, and diverse research initiatives.

The emergence of big data technologies has enabled researchers to process larger volumes of information than ever before, making this wealth of conversational data accessible to Machine Learning models and Natural Language Processing research. These analytical tools and data sources have allowed researchers to significantly expand both the volume and quality of their research.

1.2 Motivation and Need

In recent years we have witnessed a trend among social networks, forums, and instant messaging applications to restrict access to private data on their platforms(Tromble, 2021). Notable examples of this phenomenon include Reddit's denial of access to crawlers, preventing researchers' ability to collect data from its forums, and X (formerly Twitter) implementing paid API access that has severely limited research capabilities(Murtfeldt et al., 2024). For instant messaging applications, no APIs currently exist that allow collecting private conversational data, creating substantial gaps in researchers' ability to study authentic communication patterns.

This widespread restriction has created an urgent need to develop alternative solutions that will allow researchers to regain ethical access to private conversational data.

To preserve these principles, researchers require robust anonymization capabilities, particularly for language-specific research. Since studies using this platform are designed to work with Hebrew conversational data, there was a specific need to identify and implement tools that could provide effective anonymization components for Hebrew language processing.

1.3 Client and Target Audience

The main purpose of this project is to help researchers collect textual private data from users who have agreed to participate and donate selected conversational data for research purposes.

1.3.1 Primary Client

This platform is primarily intended to be used by academic researchers to collect textual data across various research fields. Survey institutes can also leverage this tool to conduct comprehensive research studies.

1.3.2 Target Audience

The platform is designed to be accessible to the general public, specifically addressing users who communicate in Hebrew on one of three supported messaging platforms: WhatsApp, Telegram, and Signal. These participants represent the data contributors who voluntarily share selected conversations for research purposes.

1.4 Innovation

The absence of APIs for private messaging data led to developing this tool. The platform addresses a gap in data collection for Hebrew language research, as no existing data collection tools offer anonymized Hebrew text .

1.5 General Solution Description

The platform is accessible through a web application. Once a user registers, they can choose to donate chats from one or more instant messaging platforms by scanning QR codes for each desired platform. After the QR code is scanned, the user's account is connected to the Matrix server and bridge system. Users can then select specific chats they wish to donate through the web application interface, ensuring that only explicitly chosen conversations are included in the research data. The platform monitors messages exclusively in the user-selected chats. All monitored conversations pass through an anonymization component. Following anonymization, the processed text is written to flat files which can be exported by researchers for analysis.

1.6 Challenges

The development of this project presented several implementation challenges that required technical solutions and workflow adaptations.

The initial phase involved establishing the matrix server infrastructure, which proved challenging due to the complexity of the matrix server architecture. This was resolved through thorough study of server documentation and configuration requirements.

Once operational, we faced integration challenges between the web application and the matrix server system. This required developing mechanisms for QR code generation and transmission, implementing efficient polling systems to monitor chat activities, and ensuring proper coordination between system components through the matrix bridges.

A significant practical challenge emerged mid-year with Google Cloud Platform resource management. We discovered that GCP charged our account even when VMs were down, resulting in depleted credits and lost resources. This led to several weeks without server access, forcing us to adapt our development workflow to focus on client-side components until additional credits were secured.

1.7 Document Structure

This document is structured as follows:

Section 2 - Researching the Existing Situation

Section 3 - Solution Description
Section 4 - System Characterization
Section 5 - Development Methodology
Section 6 - Description of Tests
Section 7 - Summary
The document concludes with a complete bibliography and appendices containing detailed technical diagrams, supplementary documentation, and web application screenshots.

2 Researching the Existing Situation

2.1 Current Survey Methodologies

Modern survey research follows established methodologies that have evolved from traditional approaches to incorporate digital technologies, though several persistent challenges continue to affect data quality and research effectiveness.

2.1.1 Traditional Survey Workflow

Traditionally survey institutes are expected to follow an established workflow: defining survey objectives, identifying target populations, selecting survey methods (phone, online, or face-to-face), designing relevant questions, choosing sampling methods and sample sizes, monitoring response rates, processing and analyzing data, and presenting findings through reports and publications (Kelley et al., 2003).

2.1.2 The Digital Transformation of Data Collection

The emergence of social networks and instant messaging applications fundamentally transformed data collection methodologies in the early 21st century. Platforms like Facebook, Twitter, WhatsApp, and Telegram created unprecedented opportunities for researchers to access authentic, real-time communication data at scale:

Expanded reach and accessibility: Digital platforms enabled researchers to access geographically dispersed populations that were previously difficult to reach through traditional survey methods, significantly broadening sample diversity.

Real-time data collection: Unlike traditional surveys that captured point-in-time responses, social media and messaging platforms offered continuous streams of naturally occurring conversations, allowing researchers to observe opinion formation and evolution in real-time.

Reduced artificial context: Communication on these platforms represented authentic exchanges rather than responses to researcher-designed questions, potentially reducing social desirability bias and artificial response patterns.

Automated analysis capabilities: The digital nature of these communications enabled computational approaches to data analysis, including natural language processing and network analysis at previously impossible scales.

This digital transformation initially promised to address many limitations of traditional survey methods by providing access to natural communication patterns. However, as privacy concerns increased and platforms implemented stricter data access policies, researchers began facing a new set of challenges in accessing and utilizing these valuable data sources.

2.2 Existing Platforms and Their Limitations

Several platforms currently address data collection from social networks, each with distinct approaches and limitations:

Amazon Mechanical Turk (mturk): A crowdsourcing platform that enables researchers to collect private chat data from users in exchange for payment. While it provides access to a large, geographically diverse user base, it faces limitations including demographic bias toward income-seeking users, restricted geographic availability, and concerns about data authenticity since participants can potentially alter information (Amazon, 2024).

The National Internet Observatory (NIO): Collects data from computers, tablets, and mobile phones through browser extensions or mobile applications, either voluntarily or for payment. The platform is restricted to American participants aged 18 and older and depends on an existing user database, limiting global accessibility. Additionally, collected data does not undergo anonymization (The National Internet Observatory, 2024).

Crimson Hexagon: A distributed system designed for secure automated data collection through background applications on mobile devices(Etlinger & Amand, 2015). While it ensures privacy without accessing personal content, the system cannot access actual message content (such as WhatsApp messages), limiting data types to metadata like time, frequency, and IP addresses rather than conversational content.

2.3 Platform Comparison and Identified Gaps

The following comparison illustrates the key features across existing platforms and our proposed Vox Populi Platform:

Feature	**VPP**	**mturk**	**NIO**	**CH**
Access to private chat content	Yes	Yes	Yes (non-VPP only)	No
Scalability	Yes	Yes	Yes	Yes
Advanced technology (Client side)	Yes	No	No	Yes
Advanced technology (User side)	No	No	No	No
Support for Hebrew	Yes	Yes (limited)	No	No
Automated data pulling	Yes	No	Yes	Yes
Data reliability and integrity	Yes	Depends on participant	Yes	Yes
Seamless Data Processing	Yes	No	No	Yes
Anonymization	Yes	No	No	Yes

Table 1: Comparison of Data Collection Platform Features

Note: VPP = Vox Populi Platform, NIO = National Internet Observatory, CH = Crimson Hexagon

2.4 Research Gaps and Platform Justification

The analysis reveals several critical gaps in existing data collection platforms that justify the development of the Vox Populi Platform:

Hebrew Language Processing: No existing platform provides Hebrew text anonymization capabilities, creating a significant gap for Hebrew-speaking research populations.

Authentic Conversational Data Access: While platforms like Crimson Hexagon maintain privacy, they cannot access actual message content, limiting research to metadata analysis rather than authentic conversational patterns.

User-Controlled Data Sharing: Current platforms lack granular user control mechanisms that allow participants to selectively share specific conversations while maintaining privacy over others.

Integrated Anonymization: Most platforms require separate processing steps for anonymization, if available at all, creating additional workflow complexity and potential privacy risks.

Multi-Platform Integration: No existing solution provides seamless integration across multiple messaging platforms (WhatsApp, Telegram, Signal) with unified data processing capabilities.

These identified gaps demonstrate the need for a specialized platform that combines authentic conversational data access, Hebrew language processing, user-controlled data sharing, and integrated anonymization capabilities—precisely what the Vox Populi Platform addresses.

3 Solution Description

3.1 General Description of the Vox Populi Platform

This project represents a solution designed to address the gap in collecting authentic conversational data for research purposes while maintaining privacy and ethical standards. The platform's architecture centers around providing researchers with access to real-world Hebrew conversational data through a user-controlled, privacy-preserving system. Since the data is contributed voluntarily it was important to make the platform accessible through open-source code, to build trust with contributors by being transparent with how the data is collected.

3.1.1 Platform Overview

Vox Populi Platform operates as a web-based application that enables participants to selectively share conversations from messaging applications for research purposes. The core of the platform is actually the matrix server,which upon setting it up and setting it's bridges up the system supports three messaging platforms: WhatsApp, Telegram, and Signal. The application integrates with the bridges, enabling contributors to connect their accounts and selecting conversations to contribute. The platform then monitors those chats, applies anonymization to ongoing text, and writes the anonymized text to flat files.

The platform's core functionality revolves around establishing secure connections between users' messaging accounts and a centralized research infrastructure through the Matrix protocol. This approach ensures that participants maintain complete control over their data sharing decisions while providing researchers with access to authentic conversational patterns.

3.1.2 User Experience and Data Flow

The platform was initentionally developed as an open-source code to show to potential contributors how the system is working and allow them to make informed decision whether they are willing to contribute.

The user interaction begins with registration through the web application interface. Following registration, participants can choose to connect one or more of their messaging platforms by scanning QR codes generated specifically for each platform.

Once connected, the system displays all available conversations (referred to as ”rooms” in Matrix terminology) from the connected messaging platforms. Participants can then make granular decisions about which specific conversations they want to include in research data collection. This selective sharing mechanism ensures that users retain full autonomy over their privacy while contributing to research initiatives.

The platform continuously monitors messages exclusively from user-selected conversations. All collected data immediately passes through an anonymization pipeline before being made available to researchers, ensuring that no personally identifiable information is accessible through the research interface.

It is possible for contributors to stop contributing when they feel uncomfortable, they are capable of disconnecting their account directly from their private cellphone device through one of the applications.

3.1.3 Data Processing and Anonymization

The platform's anonymization component is designed as a modular system where the code uses the anonymizer as a component that receives text and outputs the anonymized version. This approach makes the component easily replaceable, allowing for different models or anonymization algorithms to be integrated as needed.

The anonymization process operates in real-time, ensuring that no unprocessed personal data is stored within the system.

3.2 Matrix Protocol Approach

The Matrix protocol was selected as the foundation for the platform due to its decentralized architecture which enables secure integration with multiple messaging platforms simultaneously through standardized bridges, eliminating the need for developing custom connections to each messaging service. The platform operates a dedicated Matrix server as the central hub for all communication routing and data processing. This isolated server approach provides several benefits for research purposes: it separates research participants from the public Matrix network, enables controlled monitoring of only contributed messages, and ensures consistent data processing across different messaging platforms.

3.3 Reasons for Choosing This Solution

3.3.1 Technical Justification

The selection of the Matrix protocol as the platform's foundation was driven by several critical technical requirements that alternative approaches could not adequately address.

Multi-Platform Integration: Traditional approaches to messaging data collection require separate integration efforts for each messaging platform, often involving complex reverse engineering of proprietary protocols or reliance on unofficial APIs that may be discontinued without notice. The Matrix bridge ecosystem provides standardized, well-maintained connections to major messaging platforms through a unified protocol interface.

Scalability and Maintenance: Direct integration with messaging platform APIs presents significant maintenance challenges as platforms frequently update their protocols, potentially breaking custom integrations. Matrix bridges are maintained by dedicated development communities that ensure continued compatibility, reducing the platform's long-term maintenance burden.

Security and Compliance: Matrix's federated architecture and encryption-first design align with privacy requirements for handling personal communication data. The protocol's transparency and open-source nature enable security auditing and compliance verification that would be difficult with proprietary solutions.

3.3.2 Ethical and Privacy Considerations

The Matrix-based approach addresses critical ethical considerations that influenced the platform's design:

User Autonomy: The platform's design ensures that users maintain complete control over their data sharing decisions. Unlike approaches that require wholesale access to messaging accounts, the Matrix bridges enables granular conversation-level sharing permissions. Moreover, contributors are able to disconnect their accounts at any moment and stop contributing immediately.

Transparency: The open-source nature of Matrix components enables participants to understand exactly how their data is being processed and transmitted, supporting informed consent principles essential for ethical research.

3.3.3 Research Requirements Alignment

The chosen solution directly addresses the specific requirements identified through analysis of existing platforms and research needs:

Hebrew Language Support: Unlike existing platforms that provide limited or no Hebrew language processing capabilities, the platform was designed specifically to handle Hebrew conversational data with specialized anonymization tools.

Authentic Data Access: The platform provides access to actual conversational content rather than metadata alone, enabling research into communication patterns, linguistic phenomena, and social dynamics that require full message content analysis.

Research Workflow Integration: The platform's flat file-based exports make the data accessible to researchers, eliminating the need for complex data transformation processes.

4 System Characterization

4.1 System Architecture Overview

Vox Populi Platform implements a distributed architecture centered around the Matrix protocol as its foundational communication layer. The system consists of four main components:

1. **Matrix Server Infrastructure:** A dedicated Matrix homeserver (Synapse) with bridges to WhatsApp, Telegram, and Signal.

2. **Web Application Interface:** A Streamlit-based user interface for both research participants and researchers.

3. **Anonymization Pipeline:** A multi-stage processing system for Hebrew text anonymization.

4. **Matrix Client Code Implementation:** A Go-based client that connects the Matrix protocol infrastructure to the message processing pipeline.

The complete system flow is illustrated in Figure 1, and the database structure is shown in Figure 2.

4.2 Key Components and Technology Stack

The platform utilizes several technologies across its components:

Infrastructure and Deployment:

- Google Cloud Platform for hosting and service infrastructure
- Terraform for Infrastructure as Code
- Ansible for automated deployment and configuration
- Docker for containerization and service isolation

Communication and Data Processing:

- Matrix Protocol (Synapse homeserver) for messaging and bridge communication
- PostgreSQL for structured data storage
- Caddy web server for secure HTTPS access
- Google Gemini API for text anonymization

User Interface:

- Streamlit for web application development
- Python-based asynchronous communication with Matrix server

4.3 Matrix Protocol

The Matrix protocol serves as the foundational communication layer for the Vox Populi Platform, providing a decentralized, open-source framework for real-time communication and data synchronization.

The protocol operates as a federated communication network that enables interoperability between different messaging services while maintaining user control over data and privacy. The protocol's decentralized nature allows for distributed server infrastructure where no single entity controls the entire communication network (Li et al 2023).

External communication services is available through bridge applications. The protocol's modular architecture allows bridges to translate between Matrix's native communication format and platform-specific protocols, enabling seamless integration with WhatsApp, Telegram, Signal, and other messaging services.

For detailed information on Matrix Protocol implementation, see Appendix C.1.

4.4 Web Application

The user interface is implemented as a Streamlit web application that serves as the primary interaction point for both users and researchers. The application provides intuitive interfaces for account management, chat donation, project participation, and research data analysis.

The main application entry point is implemented in `app.py`, which establishes the foundational structure for user authentication and role-based access control (see Figure 3). The application utilizes Streamlit's session state management to maintain user authentication status and implements a clean separation between user and researcher interfaces.

For detailed information on Web Application implementation, see Appendix C.2.

4.5 Data Anonymization

The anonymization component addresses the critical challenge of protecting participant privacy while preserving the linguistic and contextual value of Hebrew conversational data for research purposes. The system must handle multiple types of sensitive information including personal names, contact details, identification numbers, URLs, and other personally identifiable information embedded within natural Hebrew text.

The platform implements an API-driven anonymization approach using Google Gemini, leveraging cloud-based AI for privacy-preserving text processing(The Gemini Team, 2023). This approach was selected for its development efficiency, serverless architecture benefits, dynamic resource allocation, and cost optimization compared to locally-hosted DictaBert Large NER which cosumes GPU.
For detailed information on Anonymization implementation approaches and technical details, see Appendix C.3.

4.6 Infrastructure and Deployment

The Platform is deployed on Google Cloud Platform (GCP), leveraging multiple cloud services to provide a scalable and reliable infrastructure. The deployment is managed through Terraform for infrastructure definition and Ansible for service configuration, with Docker providing containerization for all platform services.
For detailed information on infrastructure components and deployment architecture, see Appendix C.4.

5 Development Methodology

5.1 Development Approach

The Platform development followed a component-based modular development methodology, with parallel development tracks that later converged for system integration. This approach was selected to optimize development efficiency and align with the platform's architecture, which consists of three primary components: the Matrix server infrastructure, the anonymization pipeline, and the web application interface.

5.1.1 Development Phases and Iterations

The project development was structured in three distinct phases across two academic semesters:
Phase 1: Parallel Component Development (First Semester) During the first semester, the team initiated concurrent development tracks for two critical system components:

- *Matrix Server Development:* This track focused on establishing the Matrix protocol infrastructure, including server configuration, bridge deployment, and messaging platform integration.
- *Anonymization Pipeline Development:* Simultaneously, work began on the anonymization component, including research into Hebrew NER models, implementation of pattern-based detection systems, and testing of anonymization approaches.

Phase 2: Web Application Development (Early Second Semester) As the Matrix server infrastructure was operational, development resources shifted toward the web application component:

- *User Interface Design:* Development of the Streamlit-based web application began with user interface prototyping and workflow mapping for both participant and researcher roles.
- *Database Integration:* Concurrent with UI development, the team implemented database schemas and integration components to support user management and research project coordination.

Phase 3: System Integration and Refinement (Late Second Semester) The final development phase focused on integrating all components into a cohesive platform by integrating the conmponents and testing the platform.

5.2 Code Management, Version Control, and Documentation

The project implemented a distributed repository architecture using GitHub for version control and code management. The codebase was organized into four distinct repositories, each corresponding to a major system component(see Appendix E for repository links):

1. **GCP Resources and Terraform Repository:** This repository contained all infrastructure-as-code components, including Terraform configurations, Google Cloud Platform resource definitions, and deployment scripts.
2. **Matrix Server Repository:** This repository housed all Matrix server configurations, Ansible playbooks, Docker compose files, and related server-side components.
3. **Web Application Repository:** The Streamlit-based user interface code, database integration components, and researcher tools were maintained in a dedicated repository.
4. **Go Matrix Client Repository:** The client-side Matrix integration code written in Go, including the anonymization pipeline integration, was maintained in its own repository.

This distributed repository approach provided several advantages for the project's development workflow:

Component Isolation: Each major system component maintained its own version history, issue tracking, and release cycles. This isolation prevented changes in one component from affecting the stability or development velocity of other components.

Targeted Continuous Integration: Each repository could implement customized continuous integration workflows appropriate for its specific technology stack and testing requirements.

Collaborative Documentation: The GitHub-based documentation approach enabled distributed team collaboration, ensuring accuracy and completeness of technical documentation.

5.3 Challenges and Adaptations

Throughout the development process, the team encountered several significant challenges that required methodological adaptations and technical pivots.

5.3.1 Technical Expertise Limitations

Challenge: The team identified a significant knowledge gap in traditional web development technologies, including HTML, CSS, and JavaScript frameworks. This knowledge limitation posed a potential barrier to developing the user-facing components of the platform, which required both researcher and participant interfaces.

Adaptation: Rather than investing substantial time in acquiring traditional web development skills, the team strategically adopted Streamlit as the primary web application framework. This Python-based framework enabled rapid development of interactive web interfaces without requiring extensive HTML, CSS, or JavaScript knowledge. The Streamlit approach allowed the team to leverage their existing Python knowledge while still delivering a functional and responsive web application.

5.3.2 Infrastructure Stability and Resource Management

Challenge: At the beginning of the second semester, the team unexpectedly lost access to Google Cloud Platform virtual machines without prior notification. This loss disrupted the development timeline and threatened progress on server-dependent components. The lack of advance warning indicated insufficient monitoring and resource management procedures.
Adaptation: The team implemented an Infrastructure as Code (IaC) approach using Terraform to systematically define, provision, and manage all cloud resources. The Terraform implementation ensured that all infrastructure components were explicitly defined, making the environment reproducible and providing greater visibility into resource allocation and utilization. This approach significantly improved the team's ability to recover from infrastructure disruptions and maintain consistent development environments.

5.3.3 Budget Constraints and Resource Availability

Challenge: Several weeks into the second semester, the team exhausted their allocated Google Cloud Platform credits, rendering the Matrix server environment inoperable. This unexpected resource constraint threatened to halt development on server-dependent components and potentially jeopardize the entire integration timeline.
Adaptation: The team implemented a parallel development strategy that decoupled web application development from server availability. While awaiting resolution of the GCP credit issue, development efforts were redirected toward: enhancing the web application's user interface and interaction flows and implementing mock data services to simulate server responses.
This adaptation demonstrated the flexibility of the component-based development approach, allowing productive work to continue on client-side components despite server unavailability. When additional credits were eventually secured, the parallel development efforts could be quickly integrated with the restored server infrastructure.

6 Description of Tests

The testing strategy for the Vox Populi Platform primarily centered on a user-focused approach, culminating in a controlled pilot study with 10 participants who donated selected conversations from their messaging applications. This methodology enabled evaluation of the platform under authentic usage conditions while gathering valuable user feedback. Prior to the pilot implementation, the system underwent technical testing to ensure platform stability, functionality, and performance during the pilot phase.

6.1 Pre-Pilot Technical Testing

Prior to conducting the user pilot, comprehensive technical testing was performed to identify and resolve potential issues that could impact user experience or data integrity. This testing phase was critical for ensuring that all system components functioned correctly both independently and as an integrated solution.

6.1.1 Component Testing

Multiple test scenarios were designed and executed to evaluate distinct aspects of the platform:

- **QR Code Generation and Authentication:** The Matrix bridge connections were systematically tested to verify reliable QR code generation across all supported messaging platforms (WhatsApp, Telegram, Signal). Testing included verification of code expiration behaviors, scanning outcomes, and connection establishment reliability.
- **Web Application Functionality:** The Streamlit-based interface underwent functional testing across all user workflows including registration, login, platform connection, chat selection, and donation management. This testing verified appropriate response handling, error messaging, and user feedback mechanisms.
- **Database Integration:** User actions were tracked through the system to ensure proper reflection in the PostgreSQL database. Tests validated that chat donation selections, user registrations, and platform connections were accurately recorded and persisted across sessions.
- **Matrix Client Code:** The Go-based client implementation was tested for proper handling of user connections, message retrieval, and data processing.
- **Anonymization Component:** The text anonymization component underwent verification using sample Hebrew conversations containing various types of personally identifiable information. Testing confirmed the component worked while it was capable of handling names, addresses, phone numbers, and other sensitive data types in Hebrew textual contexts.

6.1.2 Integration Testing

Beyond individual component validation, integration testing assessed how effectively the system operated as a cohesive platform. These tests focused on data flow between components, examining how information traveled from messaging platforms through the Matrix bridges, into the anonymization pipeline, and ultimately to the research data storage.

Key integration points tested included:

- Connection between the web application and Matrix server for user account management
- Message flow from the Matrix client to the anonymization pipeline
- Database updates triggered by user actions in the web interface

6.2 User Pilot Study

Following the testing phase, a controlled pilot study was conducted with 10 participants to evaluate the platform under real-world conditions and gather user feedback on functionality, usability, and overall experience.

6.2.1 Pilot Design

The pilot study was structured as a 72-hour evaluation period, providing sufficient time to observe both initial onboarding experiences and sustained platform usage patterns. Multiple aspects were tracked:

- User connectivity status and session persistence
- Message processing volumes and completion rates
- Processing times for donated chats

Participants were instructed to connect at least one messaging platform to the system and donate as many conversations as they felt comfortable sharing during the pilot period. Participants were encouraged to use the platform naturally and report any issues or observations, enabling real-time problem identification and resolution.

6.2.2 Pilot Results and Findings

The pilot study yielded valuable insights into both technical performance and user experience aspects, highlighting both system strengths and areas requiring further refinement.
Key Technical Insights:

- **Onboarding and QR Code Generation:** All 10 participants successfully completed the onboarding process, demonstrating the effectiveness of the account creation workflow. One participant required assistance with the WhatsApp QR code generation and scanning process,
- **Chat Donation and Monitoring:** All users successfully donated chats through the platform interface. The messages were monitored properly.
- **Server Latency Variations:** Significant processing time variations were observed during the pilot. Some donated chats were quickly anonymized and written to GCP buckets, while others experienced delays of several hours before appearing in storage.
- **Web Application Impact on Server Resources:** Excessive use of interactive elements such as ”refresh my chats,” ”login,” and ”save changes” buttons occasionally resulted in server timeouts for affected users.

The pilot successfully validated the platform’s core functionality while identifying specific areas for optimization before full-scale deployment. These findings directly informed subsequent development priorities.

6.3 Infrastructure Cost Analysis

An important aspect of platform evaluation involved analyzing the operational costs associated with the Google Cloud Platform infrastructure. This analysis was critical for understanding the financial sustainability of the platform for ongoing research operations.

Service Description	**Cost (NIS)**
Cloud Run	21.11
VM Manager	4.43
Compute Engine	56.21
Cloud Storage	0.03
Cloud SQL	12.72
Vertex AI	3.9
Networking	4.54
Secret Manager	0.75
Cloud DNS	0.26
Artifact Registry	2.5
Total	**106.45**

Table 2: Google Cloud Platform Service Costs

The cost analysis revealed that Compute Engine resources constituted the largest expense (52.8% of total costs), primarily due to the continuous operation of the Matrix server and bridge virtual machines. Cloud Run (19.8%) and Cloud SQL (11.9%) represented the next largest cost categories, supporting the web application and database operations respectively.

6.4 User Feedback Analysis

To evaluate the user experience and gather structured feedback, participants completed a Google Forms questionnaire following their participation in the pilot.

6.4.1 Survey Methodology

The post-pilot survey primarily employed quantitative questions with additional open-ended questions for qualitative feedback. The questionnaire addressed several key dimensions of both technical experience and user perceptions:

- Ease of QR code generation
- Clarity of the chat donation process
- Ease of stopping donation and disconnecting from the platform
- Interface usability and navigation
- Confidence level of the participants when donating chats
- Impact of the chat selection feature on participants' willingness to donate conversations
- Likelihood of continued participation in future research
- Influence of personal acquaintance on the decision to participate
- Presence of moral dilemmas experienced during participation

Quantitative questions utilized a 5-point Likert scale (1-Strongly Disagree to 5-Strongly Agree), while open-ended questions provided opportunities for detailed feedback and improvement suggestions.

6.4.2 Key Findings

Analysis of survey responses revealed several important insights into the user experience:

Quantitative Results:

- **Ease of Use:** User responses indicated that the web application was convenient to use. When asked about the connection process, 85.7% of participants strongly agreed that it was easy to connect, while the remaining 14.3% agreed. Regarding the chat selection process, 85.7% strongly agreed it was quick and easy. For disconnecting from the platform, 57.1% strongly agreed it was easy and clear, while 42.9% agreed.
- **Privacy Protection Confidence:** Responses in this category were more varied. Only 28.6% of participants strongly agreed they felt confident giving limited access to their chats, while 71.4% somewhat agreed. When asked if the chat selection process gave them confidence to participate in the pilot, 57.1% strongly agreed, 28.6% agreed, and 14.3% somewhat agreed.
- **Likelihood of Continued Participation:** Regarding willingness to donate chats for future academic research, responses were divided equally between strongly agree, agree, and disagree at 28.6% each, with the remaining 14.3% somewhat agreeing. When asked if knowing the development team was important for their comfort in participating, 57.1% strongly agreed and 42.9% agreed.

Qualitative Feedback Themes:

- **Ethical Considerations:** One participant noted that he experienced mild moral dilemma when conversations involved third parties who had not consented to the research.

This feedback provided valuable direction for refining the platform before wider deployment, highlighting the importance of transparent privacy controls and improved user guidance. The complete survey instrument and detailed response analysis are included in Appendix D.

7 Summary

7.1 Product Description

Vox Populi Platform represents a solution to the challenge of collecting authentic conversational data for research purposes while maintaining privacy and ethical standards. Through integration with the Matrix protocol and its bridge system, the platform enables researchers to access authentic messaging data from WhatsApp, Telegram, and Signal with participant consent and control.

The platform operates as a web-based application with distinct interfaces for research participants and researchers. Participants can connect their messaging accounts, select specific conversations to contribute, and monitor their participation status. Researchers can manage projects, view anonymized data, and export processed conversations.

7.2 Implementation Status

The platform has progressed from concept to implemented system. The current implementation includes all core components necessary for ethical conversational data collection:

- **Matrix Infrastructure:** Fully operational Matrix homeserver with bridges to WhatsApp, Telegram, and Signal, deployed on Google Cloud Platform.

- **Web Application:** Complete Streamlit-based interface with user authentication, platform connection management, chat selection capabilities, and researcher tools for data access.
- **Anonymization Pipeline:** Implemented API-driven anonymization using Google Gemini 2.0, providing effective protection for Hebrew conversational data.
- **Go Client Implementation:** Functional Matrix client written in Go that connects the Matrix protocol to the message processing pipeline with appropriate concurrency and error handling.

The pilot study demonstrated the platform's capability to collect, anonymize, and store conversational data. The implementation has proven its technical viability while highlighting areas for further refinement and optimization.

7.3 Key Takeaways

Although specific challenges were detailed in Section 1.5, several valuable lessons emerged from overcoming these obstacles:

- Cloud resource management demands proactive monitoring and budget planning to prevent service disruptions.
- When selecting between technical approaches (as with our anonymization implementation), balancing effectiveness, operational simplicity, and cost efficiency often leads to more sustainable solutions than pursuing technical perfection alone.
- Early performance testing under realistic usage conditions is essential for identifying potential bottlenecks before they affect user experience.

These insights have not only informed our technical decisions but also shaped our approach to project management.

7.4 Future Plans and Recommendations

Based on pilot study findings and technical assessment, several directions for future development have been identified:

- **Platform Scalability Enhancements:** Improving load balancing and resource utilization for the Matrix infrastructure would improve performance during high-traffic periods and enable larger-scale research initiatives.
- **Advanced Anonymization Capabilities:** Continue developing more sophisticated Hebrew-specific anonymization models that better preserve linguistic context while maintaining privacy protection would enhance research value.
- **Mobile Application Interface:** Developing a companion mobile application would simplify the participant experience by reducing the friction of QR code scanning and platform connection.

The open-source nature of the platform creates opportunities for community contribution and enhancement, potentially establishing a foundation for broader research infrastructure development.

In conclusion, the platform has successfully demonstrated a viable approach to ethical conversational data collection that addresses the growing challenges of research access to authentic communication patterns.

Bibliography

1. Kelley, K., Clark, B., Brown, V., & Sitzia, J. (2003). Good practice in the conduct and reporting of survey research. *International Journal for Quality in Health Care*, 15(3), 261-266.
   https://doi.org/10.1093/intqhc/mzg031

2. Tromble, R. (2021). Where Have All the Data Gone? A Critical Reflection on Academic Digital Research in the Post-API Age. *Social Media + Society*, 7(1).
   https://doi.org/10.1177/2056305121988929

3. Murtfeldt, R., Johnson, I., Dang, V., & Starbird, K. (2024, April 10). RIP Twitter API: A eulogy to its vast research contributions. *ArXiv*.
   https://doi.org/10.48550/arXiv.2404.07340

4. Etlinger, S., & Amand, W. (2015). When Marketers and Academics Share a Research Platform: The Story of Crimson Hexagon. *Journal of Digital & Social Media Marketing*, 2(4), 361-373.
   https://www.researchgate.net/publication/272375831_When_Marketers_and_Academics_Share_a_Research_Platform_The_Story_of_Crimson_Hexagon

5. Pew Research Center. (2015). Methodology: How Crimson Hexagon Works. *Pew Research Center*, April 1, 2015.
   https://www.pewresearch.org/journalism/2015/04/01/methodology-crimson-hexagon/

6. Amazon. (2024). Amazon Mechanical Turk. Available at:
   https://www.mturk.com/help (Accessed: 09 December 2024).

7. The National Internet Observatory. (2024). Available at:
   https://nationalinternetobservatory.org/faq.html (Accessed: 09 December 2024).

8. Schipper, G.C., Seelt, R., & Le-Khac, N. (2021). Forensic analysis of Matrix protocol and Riot.im application. *Digital Investigation*, 36 Supplement, 301118.
   https://doi.org/10.1016/j.fsidi.2021.301118

9. Li, H., Wu, Y., Huang, R., Mi, X., Hu, C., Guo, S. (2023). Demystifying Decentralized Matrix Communication Network: Ecosystem and Security. In *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, Ocean Flower Island, China, pp. 260-267.
   https://doi.org/10.1109/ICPADS60453.2023.00047

10. Shmidman, S., Shmidman, A., & Koppel, M. (2023). DictaBERT: A State-of-the-Art BERT Suite for Modern Hebrew. *arXiv preprint* arXiv:2308.16687.
    https://doi.org/10.48550/arXiv.2308.16687

11. The Gemini Team. (2023). Gemini: A Family of Highly Capable Multimodal Models. *arXiv preprint* arXiv:2312.11805.
    https://arxiv.org/abs/2312.11805

12. Google Cloud. (2024). Gemini 2.0 Flash | Generative AI on Vertex AI. *Google Cloud Documentation*.
    https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash

13. Streamlit. (2024). Basic concepts of Streamlit. *Streamlit Documentation*. `https://docs.streamlit.io/get-started/fundamentals/main-concepts`

Appendices

A Detailed System Diagrams

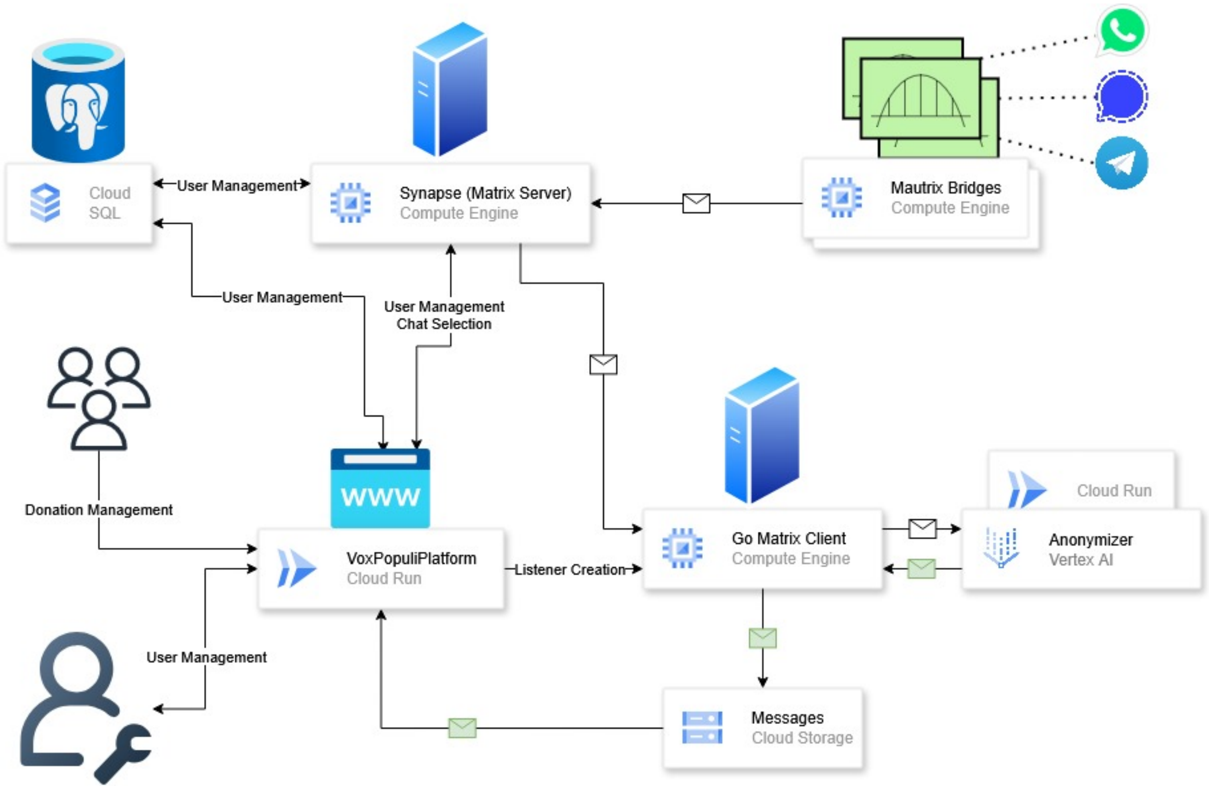
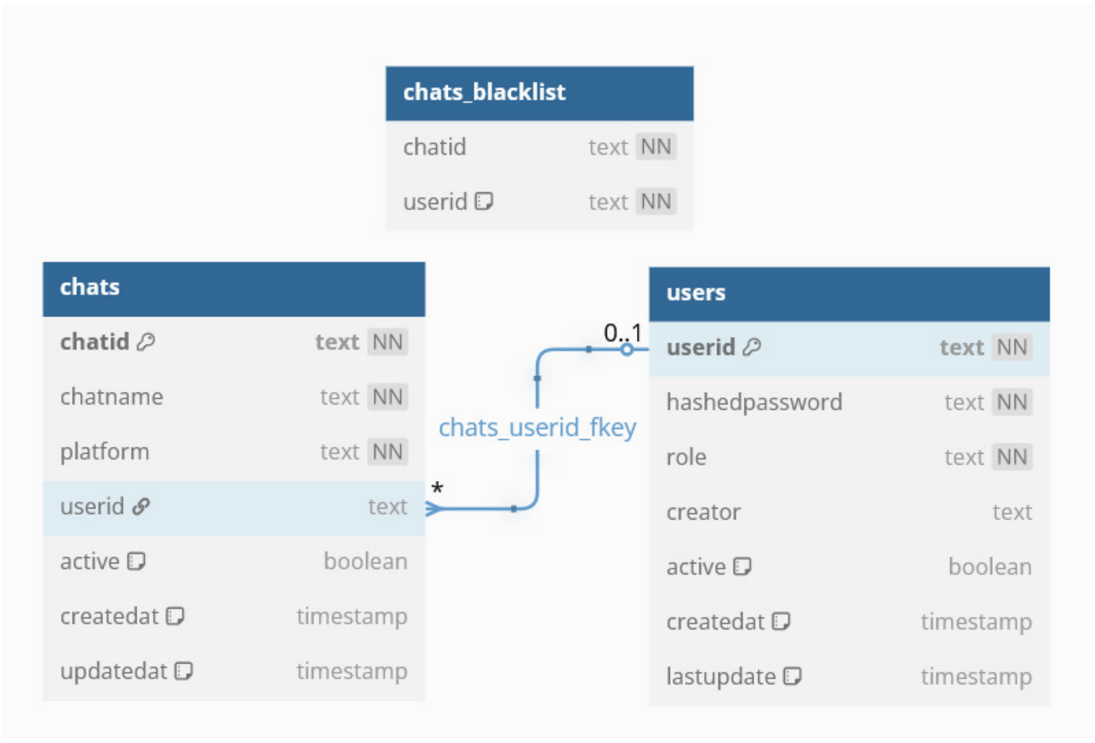


Figure 1: Vox Populi Platform System Flow Diagram



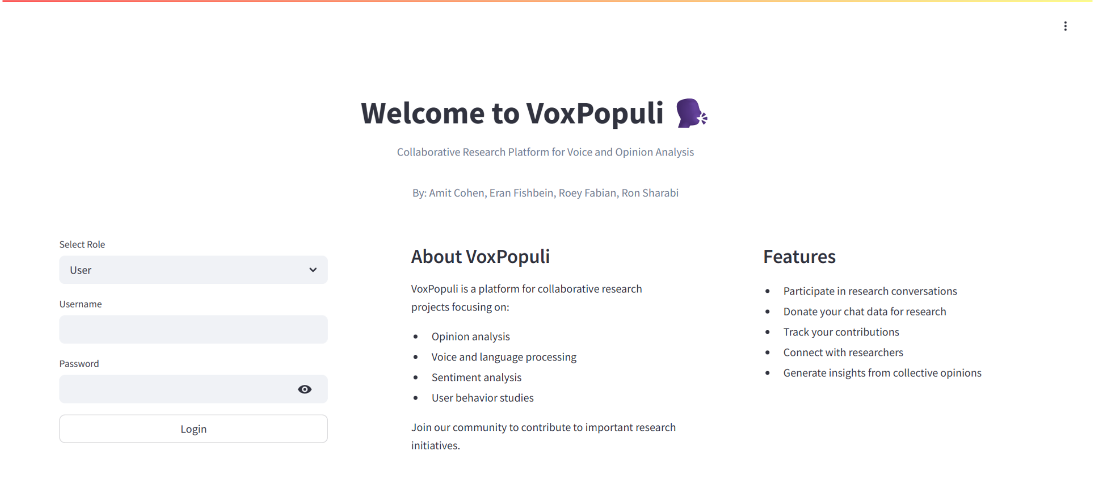
chats_blacklist		
chatid	text	NN
userid	text	NN

chats		
chatid	text	NN
chatname	text	NN
platform	text	NN
userid	text	
active	boolean	
createdat	timestamp	
updatedat	timestamp	

users		
userid	text	NN
hashedpassword	text	NN
role	text	NN
creator	text	
active	boolean	
createdat	timestamp	
lastupdate	timestamp	

Figure 2: Vox Populi Platform Database Entity Relationship Diagram

B Interface Screenshots



Welcome to VoxPopuli

Collaborative Research Platform for Voice and Opinion Analysis

By: Amit Cohen, Eran Fishbein, Roey Fabian, Ron Sharabi

Select Role: User

Username

Password

Login

About VoxPopuli

VoxPopuli is a platform for collaborative research projects focusing on:

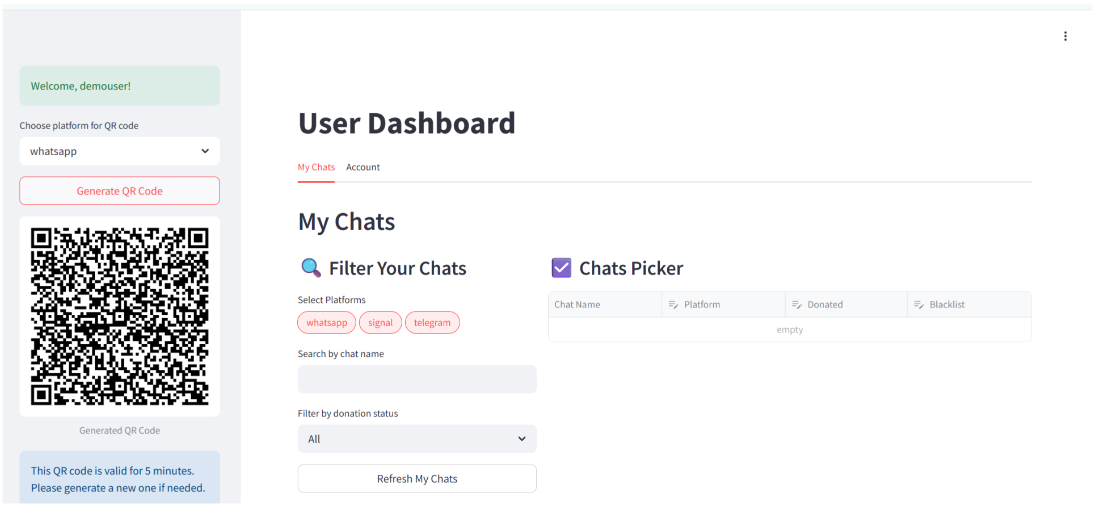
- Opinion analysis
- Voice and language processing
- Sentiment analysis
- User behavior studies

Join our community to contribute to important research initiatives.

Features

- Participate in research conversations
- Donate your chat data for research
- Track your contributions
- Connect with researchers
- Generate insights from collective opinions

Figure 3: Vox Populi Platform Welcome Page and Login Interface



Welcome, demouser!

Choose platform for QR code: whatsapp

Generate QR Code

Generated QR Code

This QR code is valid for 5 minutes. Please generate a new one if needed.

User Dashboard

My Chats | Account

My Chats

Filter Your Chats

Select Platforms: whatsapp, signal, telegram

Search by chat name

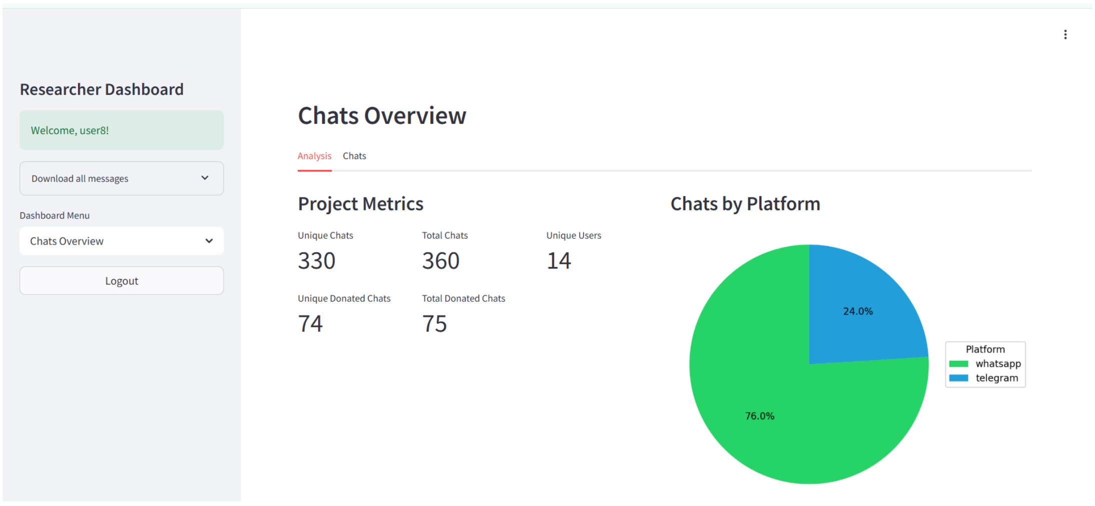
Filter by donation status: All

Refresh My Chats

Chats Picker

Chat Name	Platform	Donated	Blacklist
empty			

Figure 4: Vox Populi Platform User Dashboard Interface



Researcher Dashboard

Welcome, user8!

Download all messages

Dashboard Menu: Chats Overview

Logout

Chats Overview

Analysis | Chats

Project Metrics

Unique Chats	Total Chats	Unique Users
330	360	14

Unique Donated Chats	Total Donated Chats
74	75

Chats by Platform

Figure 5: Vox Populi Platform Researcher Interface

C Technical Implementation

C.1 Matrix Protocol Implementation

The Matrix protocol serves as the foundational communication layer for the Vox Populi Platform, providing a decentralized, open-source framework for real-time communication and data synchronization.

The protocol operates as a federated communication network that enables interoperability between different messaging services while maintaining user control over data and privacy. The protocol's decentralized nature allows for distributed server infrastructure where no single entity controls the entire communication network (Li et al 2023). This architectural approach aligns with the privacy-first requirements of research data collection, as it enables the platform to operate independently while maintaining secure connections with external messaging services.

Matrix's security model incorporates several features critical to the platform's privacy requirements. The protocol implements end-to-end encryption through the Olm and Megolm cryptographic libraries, ensuring that message content remains encrypted during transmission and storage (Schipper et al 2021).

The protocol's state-synchronization mechanisms ensure data consistency across distributed systems while maintaining cryptographic integrity. This capability is particularly important for the platform's bridge architecture, as it enables secure message routing between different messaging platforms without compromising encryption or user privacy.

External communication services is available through bridge applications. The protocol's modular architecture allows bridges to translate between Matrix's native communication format and platform-specific protocols, enabling seamless integration with WhatsApp, Telegram, Signal, and other messaging services. This extensibility is fundamental to the Vox Populi Platform's ability to collect data from multiple messaging platforms through a unified infrastructure.

The bridge ecosystem maintains Matrix's security guarantees while enabling interoperability, ensuring that the platform can access messaging data without compromising the encryption and privacy features of individual messaging platforms. The complete data processing workflow is illustrated in the System Flow Diagram.

C.2 Streamlit Web Application

The user interface is implemented as a Streamlit web application that serves as the primary interaction point for both users and researchers. The application provides intuitive interfaces for account management, chat donation, project participation, and research data analysis.

C.2.1 Application Architecture and Entry Point

The main application entry point is implemented in `app.py`, which establishes the foundational structure for user authentication and role-based access control (see Figure 3). The application utilizes Streamlit's session state management to maintain user authentication status and implements a clean separation between user and researcher interfaces.
The authentication system supports dual-role access with bcrypt password hashing for security. Upon successful login, users are redirected to role-specific applications through conditional routing, ensuring appropriate access control and functionality separation. The application maintains persistent login sessions and provides comprehensive error handling for authentication failures.
Custom CSS styling is integrated to provide a professional appearance with consistent branding, including styled headers, login containers, and responsive design elements that enhance user experience across different device types.

C.2.2 User Dashboard Implementation

The user dashboard, implemented in `user_app.py`, serves as the primary interface for research participants to manage their chat contributions and monitor their participation status. The dashboard integrates directly with the Matrix monitoring system through the `WebMonitor` class, providing real-time access to messaging platform data.

Chat Management Interface: Users can view, filter, and manage their conversations across multiple messaging platforms through an interactive data editor interface. The system displays chat information including chat names, platforms (WhatsApp, Telegram, Signal), and donation status with real-time updates from the Matrix bridges.

The chat management system implements a filtering mechanism that allows users to selectively donate conversations to research projects through a checkbox interface, with changes immediately reflected in both the local database and the Matrix server.

QR Code Generation: The application provides automated QR code generation for connecting messaging platform accounts to the Matrix infrastructure. The system implements a seven-minute cooldown mechanism to prevent excessive server requests while ensuring users can efficiently connect their accounts across multiple platforms.

QR code generation is handled asynchronously through the Matrix bridge system, with platform-specific configurations for WhatsApp, Telegram, and Signal integration. The generated QR codes are displayed directly in the sidebar for immediate scanning with mobile devices.

C.2.3 Researcher Dashboard Implementation

The researcher dashboard, implemented in `researcher_app.py`, provides comprehensive project management capabilities for research teams.

Project Analytics Interface: The researcher interface includes capabilities for analyzing collected conversational data, with integration to PostgreSQL user management databases. Researchers can view message summaries, chat statistics, interactive data tables and visualization components.

User Management System: The researcher dashboard includes comprehensive user management functionality enabling researchers to register new participants, assign users to projects, and manage participant access permissions. The system integrates with the Matrix server administration API to automatically provision user accounts and configure appropriate access controls. The user registration process includes automated server-side account creation, database record and insertion. Researchers can monitor participant status, and manage users through an intuitive interface that provides real-time feedback on registration and management operations.

Data Export and Analysis: The platform provides data export capabilities enabling researchers to extract collected conversational data for external analysis tools. The system supports filtered exports based on project criteria and time ranges while maintaining appropriate anonymization protocols.

C.2.4 Matrix Integration and Communication

The web application integrates extensively with the Matrix communication infrastructure through the `WebMonitor` and `MultiPlatformMessageMonitor` classes. This integration enables real-time communication with messaging platform bridges, user account management, and message monitoring capabilities.

Asynchronous Operations: The application utilizes Python's asyncio framework to handle Matrix server communications without blocking the user interface. All Matrix operations including login, room management, QR code generation, and message monitoring are implemented as asynchronous functions that provide responsive user experiences.

Bridge Communication: The system communicates with WhatsApp, Telegram, and Signal bridges through standardized Matrix protocols, enabling unified management of diverse messaging platforms through a single interface. The application automatically detects platform-specific rooms and applies appropriate configurations for each messaging service.

Real-time Data Synchronization: The web application maintains synchronized state between the Matrix server and local PostgreSQL databases through coordinated update mechanisms that ensure data consistency across all components.

C.2.5 Database Integration and Data Management

The application implements database integration through the `dbs.py` module, which provides object-relational mapping for PostgreSQL user management and project coordination.

Multi-Database Architecture: The system utilizes PostgreSQL for structured user, project, and relationship data.

Real-time Data Updates: The application implements real-time database synchronization ensuring that changes made through the web interface are immediately reflected in both local databases and the Matrix server infrastructure.

Data Integrity and Consistency: The system implements error handling and transaction management to maintain data integrity across database operations, with automatic rollback mechanisms for failed transactions and consistent state management across distributed components.

C.2.6 Security and Privacy Implementation

The web application implements multiple layers of security and privacy protection appropriate for handling sensitive conversational data in research contexts.

Authentication and Authorization: The system utilizes bcrypt password hashing with secure session management, implementing role-based access control that ensures users can only access appropriate functionality and data. Authentication tokens are securely managed through Streamlit's session state with automatic expiration handling.

Data Access Controls: The application implements granular access controls ensuring that users can only view and modify their own conversational data. All database operations include appropriate access control verification.

Privacy Protection: The system includes comprehensive privacy protection mechanisms including data anonymization pipelines and user consent management workflows that ensure compliance with research ethics requirements and data protection regulations.

C.3 Data Anonymization

C.3.1 Anonymization Requirements and Design Principles

The anonymization component addresses the critical challenge of protecting participant privacy while preserving the linguistic and contextual value of Hebrew conversational data for research purposes. The system must handle multiple types of sensitive information including personal names, contact details, identification numbers, URLs, and other personally identifiable information embedded within natural Hebrew text.

C.3.2 Implementation Approach A: DictaBERT Large NER

This implementation utilizes a multi-layer approach centered on Hebrew-specific Named Entity Recognition models combined with rule-based pattern detection and statistical validation.

Neural Language Model Integration: The system employs two complementary DictaBERT models specifically trained for Hebrew Named Entity Recognition: `dicta-il/dictabert-ner` (base model) and `dicta-il/dictabert-large-ner` (large model). These transformer-based models are optimized for Hebrew text processing and provide high-accuracy person name detection within conversational contexts. The system utilizes both models simultaneously to maximize recall while leveraging their complementary strengths in different linguistic contexts.

Hebrew Text Preprocessing Pipeline: Prior to NER processing, the system implements Hebrew text normalization including Unicode NFC normalization to handle character encoding variations, removal of high-plane Unicode characters that may interfere with model processing, elimination of WhatsApp-specific artifacts such as media indicators and deleted message markers, and preservation of original text structure for accurate span mapping during anonymization.

Multi-Pattern Detection System: Beyond neural NER, the system implements rule-based detection for structured sensitive information through regular expression patterns specifically designed for Israeli contexts. Phone number detection handles various Israeli mobile and landline formats including international prefixes (+972) and domestic formats. Email address detection uses comprehensive patterns that capture modern email formats. Israeli identification number detection identifies nine-digit ID numbers while avoiding false positives from other numeric sequences. Credit card and website URL detection provides additional coverage for financial and web-based identifiers.

Hebrew Morphological Variant Generation: The system addresses Hebrew morphological complexity through automated variant generation that creates all possible prefix combinations for detected names. The `generate_hebrew_variants` function applies common Hebrew prefixes) ,ל ,מ ,כש ,ש ,ו וש) to base names, generating comprehensive variant sets that capture grammatical variations of personal names as they appear in natural conversation. This approach ensures that names are consistently anonymized regardless of their grammatical context within sentences.

Statistical Validation and Filtering: To minimize false positives, the system implements probability-based validation using configurable thresholds (default 0.35) that compare the frequency of terms identified as personal names versus their total occurrence frequency in the text. Terms that appear frequently but are rarely classified as names by the NER models are excluded from anonymization, preserving legitimate Hebrew vocabulary while maintaining privacy protection.

Two-Pass Refinement System: The implementation includes a secondary anonymization pass (`second_pass_censor`) that applies the comprehensive name dictionary generated during initial processing to perform additional censoring. This second pass captures name variants and contextual appearances that may have been missed during initial NER processing, ensuring comprehensive protection while maintaining the integrity of anonymized text structure.

Overlap Resolution and Conflict Management: The system implements overlap detection algorithms that resolve conflicts when multiple entity types are detected within the same text span. Entities are prioritized by span length and entity type confidence scores, ensuring that the most comprehensive and accurate anonymization is applied while avoiding over-censoring or inconsistent masking patterns.

C.3.3 Implementation Approach B: API-Driven Anonymization with Google Gemini

The second anonymization implementation leverages Google's Gemini generative AI platform for privacy-preserving text processing. This approach, defined in `anonymizer.go`, focuses on context-aware content analysis through a cloud-based API rather than local model execution.

Prompt-Based Anonymization System: The core of this approach is a carefully engineered system prompt that instructs the Gemini model to detect and replace sensitive information in conversational text. The prompt specifically targets several categories of identifiable information:

- Personal names (including first and last names), replaced with `[NAME]`
- Address information (streets, buildings, neighborhoods), replaced with `[ADDRESS]`
- URLs and other digital identifiers, replaced with `[SPECIAL]`
- Specific identifiable dates, including Hebrew calendar references, replaced with `[DATE]`
- Financial information, passwords, and other unique identifiers, replaced with `[SPECIAL]`

Cryptographic Sender Anonymization: Independent of the Gemini API, the implementation employs SHA-256 cryptographic hashing to generate consistent pseudonyms for message senders. As shown in the `AnonymizeMessage` function, the system converts each sender ID into a deterministic 16-character pseudonym with the format `user_[hash]`, ensuring consistent anonymization across conversations.

Resilient Processing Architecture: The implementation includes error handling and retry logic through the `AnonymizeMessageWithRetry` function, which implements exponential backoff for API failures. Each API request operates under a 30-second timeout constraint, preventing processing bottlenecks.

Batch Processing and Persistence: The anonymized messages are passed to a specialized writer component that implements efficient batch processing for cloud storage. This component organizes anonymized conversations into a structured hierarchy based on username, room ID, and date.

Encrypted Message Handling: The system implements special handling for encrypted content, preserving the cryptographic status of messages while ensuring consistent processing flow. When an encrypted message is detected, the content is automatically replaced with a standard placeholder token (`[ENCRYPTED_MSG]`) without invoking the API.

C.3.4 Justification for Picking Approach B

The adoption of API-driven anonymization with Google Gemini was motivated by several strategic considerations that address both technical and operational requirements of the Vox Populi Platform.

Development Efficiency: The API-based approach drastically simplified implementation complexity. Rather than developing and maintaining complex NLP models internally, the integration required only a few concise function calls to the Gemini API. The simplicity extended to configuration management as well, with key parameters like temperature (0.1), top-p (0.95), and top-k (40) easily adjustable through the API interface without requiring model retraining or complex deployment processes.

Serverless Architecture Benefits: By leveraging Google's serverless infrastructure for text processing, the system eliminates the need for managing dedicated model servers or inference endpoints. This architecture shift removes significant operational overhead, allowing to focus on the core pipeline functionality rather than infrastructure management. The serverless paradigm also provides automatic scaling, load balancing, and fault tolerance without explicit configuration, simplifying the overall system architecture and reducing potential failure points.

Dynamic Resource Allocation: One of the most compelling advantages of this approach is the dynamic allocation of computational resources. Unlike the DictaBERT implementation which requires dedicated GPU resources regardless of workload, the API-based model scales processing capacity automatically based on current demand. During periods of low message volume, the system consumes minimal resources, while seamlessly expanding during high-traffic periods. This elasticity proved particularly valuable during pilot testing phases when message volume fluctuated significantly throughout the day.

Cost Optimization: GPU resources represent the single most expensive component in NLP processing pipelines. By offloading intensive model inference to a consumption-based API service, the system substantially reduces infrastructure costs. This pricing model proves particularly advantageous for workloads with variable volume, as it eliminates the need to provision for peak capacity that would otherwise remain idle during low-traffic periods.

C.4 Infrastructure and Deployment

C.4.1 Ansible

Ansible is an open-source automation platform that uses declarative YAML-based playbooks to define and execute infrastructure management tasks across multiple machines. The platform operates agentlessly, connecting to target systems via SSH to execute configuration management, application deployment, and orchestration tasks without requiring specialized software installation on managed nodes.

This project leverages Ansible as the primary deployment and configuration management system, orchestrating the multi-server infrastructure required for Matrix homeserver and bridge operations. The automation framework manages four distinct Google Cloud Platform virtual machines: one homeserver instance and three dedicated bridges for WhatsApp, Telegram, and Signal integration.

The platform's Ansible implementation utilizes a modular playbook architecture, with the main orchestration playbook (voxpopuli.yaml) coordinating the execution of specialized sub-playbooks that handle specific deployment phases. This modular approach ensures maintainable, reusable automation components while providing clear separation of concerns for different system configuration aspects.

The Ansible inventory defines the platform's infrastructure topology through the inventory.ini configuration file, which organizes servers into logical groups and defines host-specific variables. The homeserver group contains the primary Matrix server (vox-populi.dev), while the bridges group includes three separate virtual machines, each dedicated to a specific messaging platform integration.

Bridge servers are configured with platform-specific variables including Docker image tags, port assignments, display name formatting rules, and authentication parameters. For example, the Telegram bridge includes domain role specifications for puppeting capabilities, while the WhatsApp bridge defines custom display name templates that preserve business names and phone numbers appropriately.

Host variables specify essential configuration parameters such as root directories, configuration paths, and service ports, enabling consistent deployment across different server roles while maintaining the flexibility to customize individual service configurations based on platform requirements.

The primary deployment workflow follows a carefully structured sequence designed to ensure proper dependency resolution and system initialization. The orchestration begins with persistent disk mounting procedures, which configure and attach Google Cloud Platform persistent storage to the homeserver instance, providing durable data storage for Matrix homeserver operations.

Following storage preparation, the system generates the Synapse homeserver configuration through automated Docker container execution, creating the foundational Matrix server configuration with appropriate domain settings, database connections, and security parameters. This generation process eliminates manual configuration errors while ensuring consistent deployment across different environments.

Bridge registration represents a critical deployment phase where each messaging platform bridge generates its authentication credentials and registration files. The automation system coordinates the transfer of these registration files from bridge servers to the homeserver, automatically updating the Matrix homeserver configuration to recognize and authorize bridge connections.

Ansible's template processing capabilities enable dynamic configuration generation based on inventory variables and runtime parameters. The platform utilizes YAML processing tools (yq) within Ansible tasks to modify configuration files, merge external configuration sources, and customize service parameters based on deployment-specific requirements.

Bridge configuration management demonstrates this templating approach, where Ansible tasks dynamically configure messaging platform API credentials, database connections, homeserver addresses, and permission matrices. For Telegram bridge deployment, the system retrieves API credentials from Google Cloud Secret Manager and injects them into configuration files, ensuring secure credential management without exposing sensitive information in configuration repositories.

Database configuration integration exemplifies the platform's approach to external configuration management, where Ansible tasks merge database connection parameters from separate configuration files into service-specific configurations. This approach enables centralized credential management while maintaining configuration consistency across multiple system components.

The Ansible automation implements comprehensive error handling and idempotency mechanisms to ensure reliable deployment and configuration management. Tasks include conditional logic that prevents destructive operations on systems with existing data, such as disk formatting procedures that check for existing content before proceeding with potentially destructive operations.

Service management tasks incorporate validation checks that verify successful configuration generation before proceeding with dependent operations. For example, homeserver configuration generation includes file existence checks that prevent duplicate configuration attempts while ensuring that required configuration files are properly created.

The automation framework includes cleanup procedures that remove temporary files and reset system state appropriately, ensuring that deployment processes do not leave residual artifacts that could interfere with subsequent operations or create security vulnerabilities.

C.4.2 Docker Configuration

The Platform utilizes Docker as the primary application runtime environment, containerizing all core services including the Matrix homeserver, messaging platform bridges, and configuration management tools. This containerized architecture ensures consistent deployment environments, simplifies dependency management, and enables efficient resource utilization across the multi-server infrastructure.

Service Containerization Architecture

All platform services operate within Docker containers, providing isolation and standardization across the distributed infrastructure. The Synapse homeserver runs in the official Element HQ container (ghcr.io/element-hq/synapse:latest), ensuring access to the latest stable Matrix implementation with proper security updates and community support.

Messaging platform bridges utilize specialized Mautrix containers (`dock.mau.dev/mautrix/whatsapp`, `dock.mau.dev/mautrix/telegram`, `dock.mau.dev/mautrix/signal`). These containers include all necessary dependencies and runtime environments specific to each messaging platform's integration requirements.

The containerization approach eliminates dependency conflicts between different services while enabling independent service updates and maintenance..

Volume Management and Data Persistence

Docker volume mounting strategies ensure data persistence and enable configuration management across container lifecycles. The homeserver container utilizes two primary volume mounts: a data volume for Matrix database files and user content, and a configuration volume for homeserver settings and bridge registration files.

Bridge containers employ similar volume mounting patterns, with configuration directories mapped to host file systems to enable persistent configuration storage and external configuration management through Ansible. The volume mounting configuration uses the ":z" flag for SELinux compatibility, ensuring proper security context handling in enterprise environments.

Container Networking and Service Communication

The platform implements Docker's default bridge networking for inter-container communication, with containers accessing each other through internal IP addresses and exposed ports. The Synapse homeserver container exposes port 8008 for Matrix client-server API communication, while bridge containers expose platform-specific ports (29318 for WhatsApp, 29317 for Telegram, 29328 for Signal) for homeserver integration. Port mapping configuration enables external access to services while maintaining container isolation.

Container networking facilitates the distributed architecture where bridges operate on separate virtual machines while maintaining secure communication channels with the central homeserver. This distribution strategy provides fault tolerance and resource isolation for individual messaging platform integrations.

Container Lifecycle Management

Ansible orchestration manages Docker container lifecycles, including creation, configuration, startup, and maintenance operations. Container deployment utilizes declarative run commands that specify image sources, volume mounts, port mappings, and runtime parameters, ensuring consistent container configuration across deployments.

Container management includes cleanup procedures for temporary containers used in configuration generation, preventing resource accumulation and maintaining clean deployment environments. The automation framework tracks container states and implements appropriate lifecycle transitions based on deployment phases and operational requirements.

Security

The platform configures containers with restricted capabilities and resource limits appropriate for their operational requirements.

Container security extends to network isolation, where services communicate through defined ports and protocols while maintaining separation from host system processes. This isolation protects the host infrastructure while enabling controlled inter-service communication necessary for platform operations.

C.4.3 PostgreSQL Database

PostgreSQL serves as the primary database backend for the Matrix homeserver and bridge services, providing persistent storage for user accounts, room state, message history, and system configuration data. The database system supports the platform's requirements for real-time message processing and research data collection workflows.

The platform implements centralized database configuration through an external database.yaml file that contains connection parameters and authentication credentials shared across all platform components. This configuration file is dynamically integrated into service configurations during deployment using YAML processing tools.

For the Synapse homeserver, Ansible tasks merge the database configuration using the command: 'yq eval -i '.database = load("/app/database.yaml").database'', which loads the external database settings into the homeserver.yaml configuration file. Similarly, bridge services utilize the same centralized configuration ensuring consistent database connectivity across all services.

This centralized approach ensures configuration consistency between the Matrix homeserver and bridge services while enabling secure credential management through external configuration sources. The configuration merging process eliminates the need for duplicate database credentials across multiple service configuration files while maintaining secure access to the shared database instance.

The platform's database schema design and entity relationships are detailed in the Database Entity Relationship Diagram (see Appendix A, Figure A.2).

C.4.4 Synapse Homeserver

Synapse serves as the platform's Matrix homeserver implementation, written in Python and providing the foundational communication layer for the entire system. The Synapse server manages all Matrix protocol operations including user authentication, room management, message routing, and federation capabilities.
Key Synapse configurations implemented for the platform include:

Federation Settings: The server is configured to operate in a federated mode, enabling communication with external Matrix servers and bridge networks while maintaining security boundaries for research data.

Registration Controls: User registration is configured with specific controls to ensure only authorized research participants can access the system.

Rate Limiting: Comprehensive rate limiting configurations prevent abuse while ensuring smooth operation during high-volume data collection periods, with specific allowances for bridge traffic and automated data processing.

C.4.5 Caddy Web Server

Caddy is a modern web server that provides automatic HTTPS certificate management, reverse proxy capabilities, and simplified configuration through its Caddyfile format. The server automatically obtains and renews SSL/TLS certificates through Let's Encrypt without requiring manual certificate management.

Caddy is used in this project as the reverse proxy and external-facing web server, providing secure HTTPS access to the Matrix homeserver while handling certificate management and traffic routing. Caddy serves as the entry point for all external communications to the platform.

Reverse Proxy Configuration: Caddy is configured to reverse proxy Matrix protocol endpoints from the external domain (vox-populi.dev) to the internal Synapse homeserver running on 172.17.0.2:8008. The platform configures dual port handling with identical reverse proxy rules for both the default HTTPS port (443) and the Matrix federation port (8448). This configuration ensures compatibility with Matrix federation requirements while providing standard HTTPS access for web clients.

Automatic HTTPS and Certificate Management: Caddy automatically obtains and manages SSL/TLS certificates for the vox-populi.dev domain through Let's Encrypt integration. Certificate storage utilizes the persistent disk storage location (/mnt/disks/persistent-disk) to ensure certificate persistence across system restarts and maintenance operations. The automatic certificate management eliminates manual certificate provisioning and renewal procedures while ensuring continuous HTTPS availability for research participants.

Deployment and Configuration Management: Caddy deployment utilizes Ansible automation as the final step in the platform deployment workflow. The deployment process copies the Caddyfile configuration to /etc/caddy/Caddyfile, formats the configuration using 'caddy fmt –overwrite' to ensure proper syntax, and starts the service with 'caddy start –config

/etc/caddy/Caddyfile‘. The Caddyfile configuration is generated dynamically through Ansible templating, ensuring consistent configuration across deployments while enabling environment-specific customization.

Logging and Monitoring: The Caddy configuration includes comprehensive logging for reverse proxy operations, with log files stored at `/var/log/caddy/reverse_proxy.log`. This logging provides visibility into Matrix protocol traffic, connection patterns, and potential issues with client connectivity or homeserver communication.

C.4.6 Google Cloud Platform Infrastructure

The Platform is deployed on Google Cloud Platform (GCP), leveraging multiple cloud services to provide a scalable, secure, and reliable infrastructure for Matrix homeserver operations and messaging platform integrations.

Compute Engine Architecture: The platform utilizes Google Compute Engine to deploy four distinct virtual machines. The primary homeserver operates on an e2-medium instance (2 vCPUs, 4GB RAM) to handle Matrix protocol operations, user management, and central coordination activities. The messaging platform bridges run on e2-small instances (2 vCPUs, 2GB RAM), with each bridge dedicated to a specific messaging platform integration (WhatsApp, Telegram, and Signal).

All virtual machines utilize Debian 12 (Bookworm) as the base operating system, providing a stable and secure foundation for containerized applications. The instances are configured with automated startup scripts that handle dependency installation including Docker, user account creation, and service initialization during system boot.

Cloud SQL Database Services: The infrastructure implements PostgreSQL 16 through Google Cloud SQL as the primary database backend for Matrix homeserver and bridge operations. The database instance is configured with db-g1-small tier specifications, providing 20GB of PD-SSD storage with automatic backup and point-in-time recovery capabilities.

Database security is enforced through SSL-only connections with client certificates for authentication. The platform automatically provisions SSL certificates for each service component, ensuring encrypted communication between application services and the database instance. Backup configuration includes daily automated backups with seven-day retention and transaction log retention for point-in-time recovery capabilities.

Network Security and Firewall Configuration: The networking architecture implements a comprehensive security model through Google Cloud Firewall rules that define precise access controls for different system components. Web traffic rules permit access through ports 80 and 443 for standard HTTP/HTTPS communication, while Matrix federation utilizes port 8448 for inter-server communication according to Matrix protocol specifications.

Internal communication between bridges and the homeserver is secured through targeted firewall rules that allow bridge-to-homeserver communication on port 8008 and specific bridge ports (29318 for WhatsApp, 29317 for Telegram, 29328 for Signal). This segmented approach ensures that only authorized traffic flows between system components while preventing unauthorized external access.

DNS and Static IP Management: Google Cloud DNS manages domain resolution for both vox-populi.dev and voxpopuli.dev domains, with A records pointing to the homeserver’s static external IP address. The configuration includes subdomain support for matrix.vox-populi.dev to enable Matrix-specific routing and federation capabilities.

Static IP addresses are assigned to all instances through Google Compute Address resources, ensuring consistent external connectivity and enabling reliable DNS resolution. This approach eliminates potential connectivity issues that could arise from dynamic IP address assignment during instance restarts or maintenance operations.

Secret Management and Security: The platform integrates Google Cloud Secret Manager for secure storage and access of sensitive information including database passwords, SSL client keys, and API credentials. Service accounts are configured with appropriate IAM roles to access secrets while maintaining principle of least privilege security practices.

SSL certificate management is automated through startup scripts that retrieve certificates from Cloud SQL instances and Secret Manager, ensuring secure database connections and proper encryption for all data transmissions between system components.

C.4.7 Terraform Infrastructure as Code

Terraform serves as the Infrastructure as Code (IaC) tool for Vox Populi Platform, providing automated provisioning, configuration management, and lifecycle control of all Google Cloud Platform resources. The Terraform configuration ensures reproducible, version-controlled, and scalable infrastructure deployment.

Resource Definition and Modular Architecture: The Terraform configuration utilizes a modular approach with local variables and for_each loops to enable scalable resource creation while maintaining consistency across similar components. Bridge instances are defined through a local configuration map that specifies machine types, instance names, and startup scripts, allowing for easy addition of new messaging platform integrations without duplicating configuration code.

The main.tf file defines all infrastructure components including compute instances, networking rules, DNS records, and database configurations through declarative resource blocks. This approach ensures that infrastructure state is explicitly defined and can be consistently reproduced across different environments.

Service Account and IAM Management: Service account management is automated through Terraform resource definitions, with dedicated service accounts created for the homeserver and each bridge instance. IAM role bindings are dynamically assigned through google_project_iam_binding resources to ensure appropriate access to Google Cloud services.

The configuration implements role-based access control through predefined IAM roles including roles/secretmanager.secretAccessor for credential access, roles/cloudsql.viewer for database connectivity, roles/logging.logWriter for centralized logging, and roles/monitoring.metricWriter for system monitoring capabilities.

Network and Security Configuration: Terraform manages all networking components including firewall rules, static IP addresses, and DNS configurations through dedicated resource blocks. Firewall rules are defined with specific source and target tags that correspond to instance roles, enabling precise traffic control between system components.

The configuration includes comprehensive security measures such as SSL-only database connections, encrypted instance communications, and restricted firewall access. DNS management includes both primary domains and subdomains required for Matrix federation and web application access.

State Management and Import Capabilities: The Terraform configuration includes import blocks that enable integration with existing Google Cloud resources, providing flexibility for incremental infrastructure updates without service disruption. This approach allows the platform to incorporate pre-existing resources such as DNS zones and database instances into the managed infrastructure.

Lifecycle management rules prevent accidental destruction of critical resources such as the Cloud SQL database instance through prevent_destroy directives. Output values provide non-sensitive connection information including IP addresses and database connection details while keeping confidential data properly secured through separate secret management.

Startup Script Integration: Terraform integrates with shell startup scripts that handle service initialization and configuration management during instance boot processes. Each compute instance references specific startup scripts through metadata_startup_script parameters, enabling automated dependency installation, user account creation, and service configuration.

The startup scripts handle Docker installation, user privilege management, SSL certificate retrieval, and database configuration file generation. This integration ensures that newly provisioned instances are immediately ready for service deployment without manual configuration steps.

C.5 Client Code Implementation in Go

The client component of the Vox Populi Platform represents a critical aspect of the system's architecture, providing the interface between the Matrix protocol infrastructure and the message processing pipeline. The implementation, written in Go, was chosen for its performance characteristics, concurrency model, and extensive ecosystem of libraries.

C.5.1 Architecture Overview

The client code follows a modular design pattern with clear separation of concerns across multiple components. The core functionality is distributed across several key modules:

ClientManager: Serves as the central coordination point for managing multiple Matrix user clients. As implemented in `matrix_client.go`, this component maintains a thread-safe map of active client connections and handles user lifecycle operations including creation, authentication, and termination. The concurrent-safe design employs mutex-based synchronization to enable simultaneous operations across multiple user accounts without race conditions.

UserClient: Encapsulates the state and operations for individual Matrix accounts. Each `UserClient` instance maintains its own connection context, cancellation function, and whitelist of rooms for monitoring. This design ensures proper resource isolation between different user accounts while enabling granular control over which conversations are included in research data collection.

Message Pipeline Integration: The client implementation connects directly to the message processing pipeline through a channel-based architecture. This non-blocking approach ensures that message handling does not impact client responsiveness, while backpressure mechanisms prevent resource exhaustion during high-volume periods.

C.5.2 Matrix Protocol Integration

The implementation leverages the `mautrix` Go library to establish and maintain connections to the Matrix homeserver. This integration encompasses several key aspects:

Authentication Flow: The client implements the password-based authentication flow defined in the Matrix specification. As shown in the `CreateUser` method, the system utilizes Matrix's user identifier system and password authentication to establish connections with stored credentials for persistent sessions.

Event Subscription: The client subscribes to both standard message events (`event.EventMessage`) and encrypted message events (`event.EventEncrypted`) through the syncer registration mechanism. The implementation properly differentiates between these event types during processing, applying appropriate handling for each.

Context-Based Lifecycle Management: Each user client operates with its own context hierarchy, enabling controlled shutdown through context cancellation. This design ensures clean termination of background operations and prevents resource leaks, as demonstrated in the graceful shutdown sequence that uses context cascading to signal termination across all subsystems.

C.5.3 Concurrency Model

The Go implementation takes full advantage of the language's goroutine-based concurrency model to achieve high performance and responsiveness:

Non-Blocking Operations: All long-running operations including Matrix synchronization and message processing occur in dedicated goroutines, ensuring that control operations remain responsive regardless of processing load. The implementation carefully manages goroutine lifecycles to prevent leaks during client destruction.

Channel-Based Communication: The system utilizes Go's channel primitives for inter-component communication. This approach provides natural flow control between the client and pipeline components through channel buffering and backpressure, as seen in the message forwarding mechanism that implements non-blocking sends with appropriate error handling for buffer-full conditions.

Mutex Protection: Critical data structures including the client map and room whitelists are protected with appropriate mutex locks to ensure thread safety. The implementation utilizes read-write mutexes where appropriate to optimize for concurrent read access patterns while maintaining proper write synchronization.

C.5.4 API Interface

The client exposes its functionality through a RESTful API that enables external control and integration:

User Management Endpoints: The API provides endpoints for creating, configuring, and destroying user clients. These operations, implemented in `handlers.go`, follow RESTful conventions with appropriate status codes and error handling for robust integration with external systems.

Room Whitelisting: The API includes dedicated endpoints for setting room whitelists, enabling granular control over which conversations are monitored. This functionality is critical for the ethical operation of the platform, ensuring that only explicitly selected conversations are included in research data.

Status Monitoring: The implementation includes status reporting through both logging and API endpoints. The pipeline status handler provides real-time metrics on message processing throughput, buffer utilization, and error conditions, enabling operational monitoring and debugging.

C.5.5 Error Handling and Resilience

The client implementation incorporates error handling and resilience features:

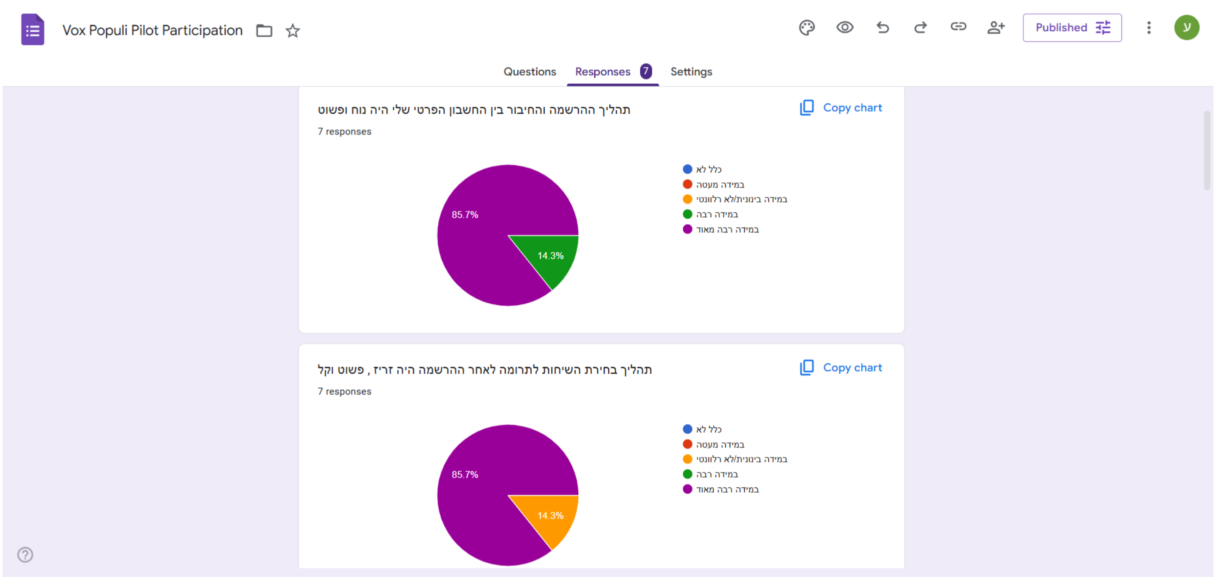
Graceful Degradation: The system handles various failure modes including network disruptions, API rate limiting, and service unavailability through appropriate retry mechanisms and fallback behaviors. The Matrix client synchronization loop includes automatic reconnection logic with exponential backoff for transient failures.

Comprehensive Logging: All operations incorporate detailed logging with appropriate severity levels, enabling troubleshooting and operational visibility. Error conditions include contextual information to facilitate rapid diagnosis and resolution.

Clean Shutdown Sequence: The implementation includes a carefully orchestrated shutdown sequence that ensures proper resource cleanup and prevents data loss during termination. This process includes cancellation propagation, drain operations for in-flight messages, and timeout-based fallback mechanisms for components that fail to terminate promptly.

The Go implementation of the client component provides a robust, high-performance foundation for the Vox Populi Platform's data collection capabilities, effectively balancing throughput, reliability, and resource efficiency while maintaining strict ethical controls over data access.

D Survey Results



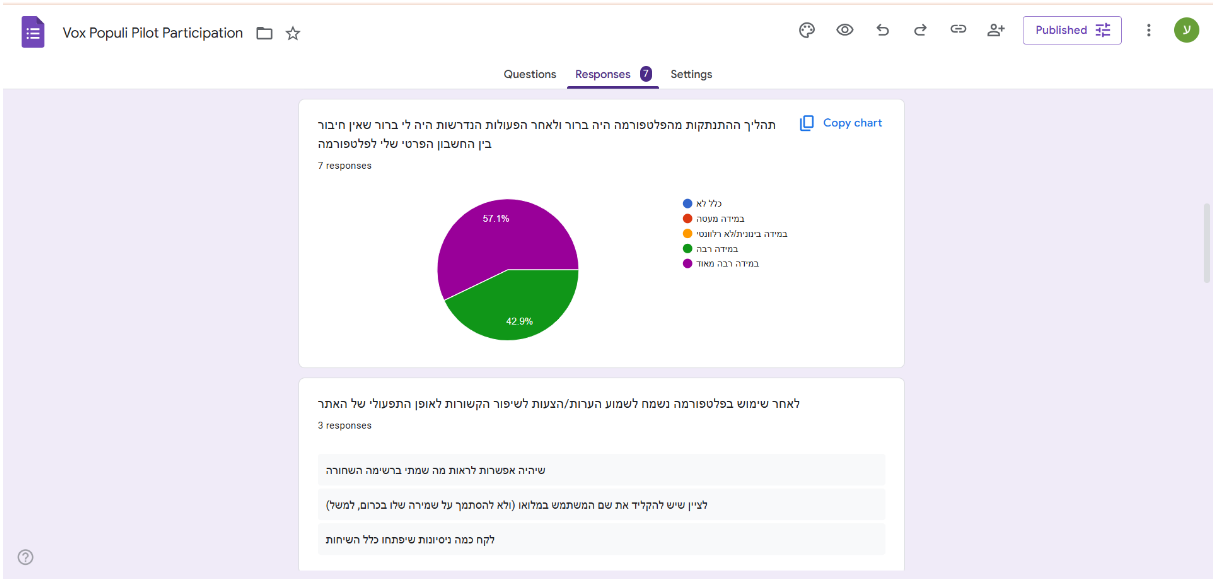
תהליך ההרשמה והחיבור בין החשבון הפרטי שלי היה נוח ופשוט

7 responses

תהליך בחירת השיחות לתרומה לאחר ההרשמה היה זריז , פשוט וקל

7 responses

Figure 6: Survey Results



תהליך ההתנתקות מהפלטפורמה היה ברור ולאחר הפעולות הנדרשות היה לי ברור שאין חיבור בין החשבון הפרטי שלי לפלטפורמה

7 responses

לאחר שימוש בפלטפורמה נשמח לשמוע הערות/הצעות לשיפור הקשורות לאופן התפעולי של האתר

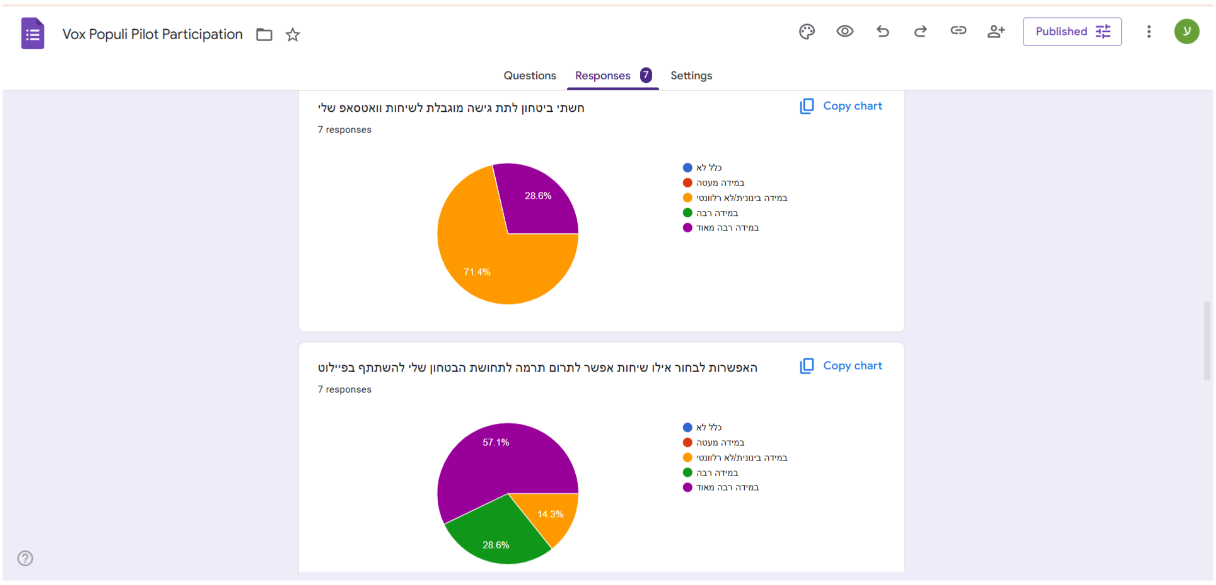
3 responses

שיהיה אפשרות לראות מה שמתי ברשימה השחורה

לציין שיש להקליד את שם המשתמש במלואו (ולא להסתמך על שמירה שלו בכרום, למשל)

לקח כמה ניסיונות שיפתחו כלל השיחות

Figure 7: Survey Result


Vox Populi Pilot Participation

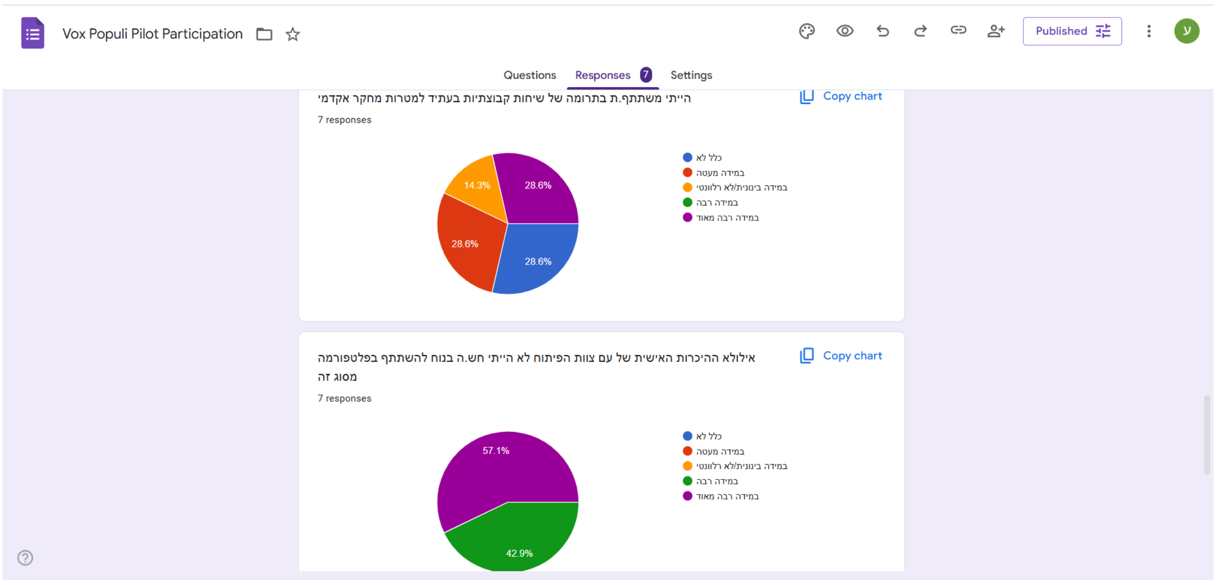
חשתי ביטחון לתת גישה מוגבלת לשיחות וואטסאפ שלי

7 responses

האפשרות לבחור אילו שיחות אפשר לתרום תרמה לתחושת הבטחון שלי להשתתף בפיילוט

7 responses

Figure 8: Survey Result


Vox Populi Pilot Participation

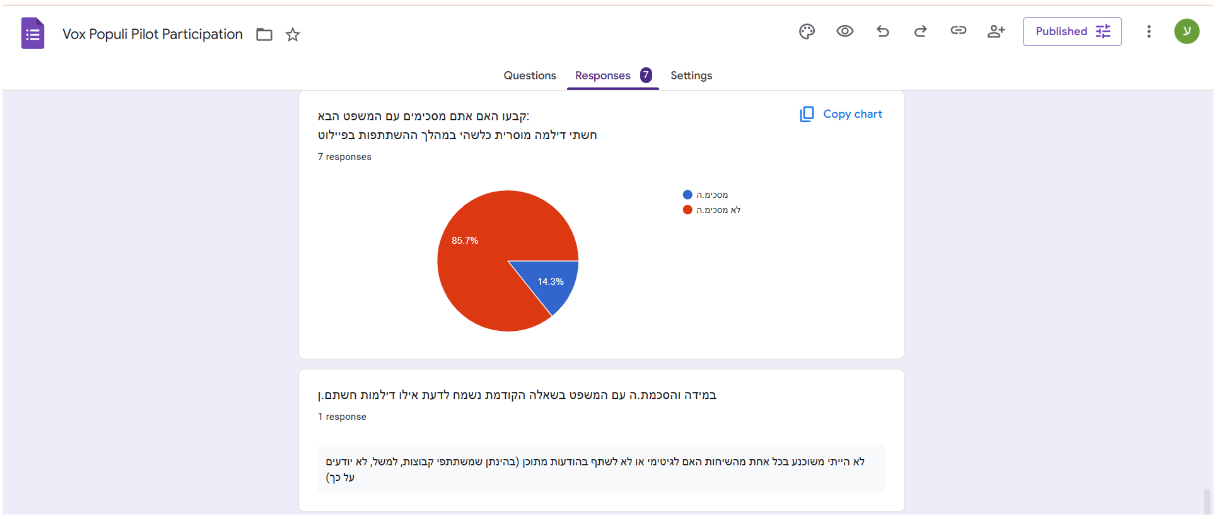
הייתי משתתף.ת בתרומה של שיחות קבוצתיות בעתיד למטרות מחקר אקדמי

7 responses

אילולא ההיכרות האישית של עם צוות הפיתוח לא הייתי חש.ה בנוח להשתתף בפלטפורמה מסוג זה

7 responses

Figure 9: Survey Result


Vox Populi Pilot Participation

קבעו האם אתם מסכימים עם המשפט הבא:
חשתי דילמה מוסרית כלשהי במהלך ההשתתפות בפיילוט

7 responses

במידה והסכמת.ה עם המשפט בשאלה הקודמת נשמח לדעת אילו דילמות חשתם.

1 response

לא הייתי משוכנע בכל אחת מהשיחות האם לגיטימי או לא לשתף בהודעות מתוכן (בהינתן שמשתתפי קבוצות, למשל, לא יודעים על כך)

Figure 10: Survey Results

E GitHub Repository Links

The project source code is available in the following GitHub repositories:

1. **GCP Resources and Terraform Repository:**
   `https://github.com/Fabian665/vox-populi-terraform`
2. **Matrix Server Repository:**
   `https://github.com/Fabian665/vox-populi-ansible`
3. **Web Application Repository:**
   `https://github.com/ronshrb/VoxPopuliPlatform`
4. **Go Matrix Client Repository:**
   `https://github.com/Fabian665/vox-populi-client`