PROJECT IN INTELLIGENT INFORMATION SYSTEMS W2024

CHATBOT AI(LLM) ASSISTANT

Amir Suleimenov K12247291

Introduction

Abstract:

Chatbots have several types and levels of complexity. They started with ELIZA, developed by Joseph Weizenbaum. ELIZA simulated conversation using pattern-matching techniques and scripts but lacked real understanding, only creating the appearance of comprehension (Weizenbaum, 1966). Based on this idea, a family of rule-based chatbots developed.
Rule-based chatbots are basic and easiest to implement. They are made to interact with users by previously determined guidelines and requirements. These solutions initiate with pre-programmed responses when they recognise certain phrases or patterns in user input. Developers manually create and design a rule-based chatbot's rules, which define how the bot will react to different user inputs. That means chatbot conversations always have strict restrictions on the chat flow algorithm. Users can't interact flexibly to get or receive information in a different order(ChatInsight 2024). In recent years, the development of neural networks and deep learning led to the creation of large language models (LLMs) such as OpenAI's GPT series. These models, trained on vast datasets and fine-tuned for natural language understanding, can engage in coherent, context-aware conversations across a wide range of topics (Brown et al., 2020). Unlike earlier models, LLMs utilize transformer architectures, allowing for superior handling of context and generation of nuanced, human-like responses (Vaswani et al., 2017). JP Morgan workers were encouraged to use LLM Suite for "writing, generating ideas, solving problems using Excel, summarizing documents," among other things, according to an email sent by the bank. JP Morgan employees were encouraged to use the LLM Suite for "writing, generating ideas, solving problems with Excel, summarizing documents," among other things, according to an email from the bank. At JPMorgan, the chatbot could augment the work with "a hybrid of human and AI analysts, similar to how the intelligence community works," Igor Jablokov, founder and CEO of AI startup Pryon told Fortune. JPMorgan has more AI chatbots in addition to LLM Suite, like Connect Coach and SpectrumGPT, which are designed for specific business tasks rather than being general-purpose tools (Confino 2024)
This adaptability makes LLM chatbots particularly well-suited for tasks like volunteer onboarding, where they can guide new users by answering questions, collecting relevant information, and recommending opportunities tailored to each user's profile and preferences.

Brief description of Project:

Chatbots have evolved significantly over the years, transitioning from basic rule-based systems to sophisticated AI-powered conversational agents. Their theoretical application for websites has expanded beyond simple user interaction, enabling a new level of engagement and personalization. In the context of modern web applications, a chatbot serves not only as a means of communication but as a core component of the user experience, offering tailored services, guiding users through tasks, and providing dynamic content based on individual needs and preferences.

The theoretical purpose of a chatbot for a website revolves around streamlining interaction, offering assistance, and improving accessibility. Websites often require systems that can handle common user inquiries, support complex workflows, and assist users in navigating the platform efficiently. A well-designed chatbot can act as a virtual assistant, guiding users through projects. Also chatbot could add specific volunteer information, troubleshooting steps, or even assist with tasks like filling out forms, providing status updates, or offering

personalized recommendations.

One of the primary purposes of a chatbot on a website is to provide real-time volunteer support. Instead of relying on static FAQs or waiting for a human representative to become available, users can get immediate answers to their questions. The chatbot can interact in a conversational manner, making the experience feel more personal and responsive. It can address a wide range of inquiries, from basic questions about the website or volunteering possibilities to more complex issues requiring the chatbot to access and integrate user-specific data, like purchase history or account status.

In this project, we document the design and development of a chatbot based on these LLM capabilities. The project contains a website, which uses Django framework and stores authorized user profiles in SQLite database with a rendering chat page. Chatbot is powered by a configured Llama LLM model, which runs through an Ollama instance. Llama 3 was chosen as its significance lies in its balance of performance, cost-efficiency, and open accessibility. Meta's experimental evaluation discovered that the Llama3 model performs on par with leading language models such as GPT-4. (Llama Team 2024). This project not only highlights the potential of LLMs in chatbots to enhance user interactions but also pushes the boundaries of what is possible with AI-driven communication.

Goals

Project Goal:

Develop an intelligent LLM chatbot assistant prototype
Measure and collect performance based on human conversations about the interaction process

System(Chatbot) Goal:
Assist web portal users(NPO managers, volunteers):
The chatbot should consult the user, provide information and help the user to use the web platform (Sub-goals achieving processes)

Sub-System Goal 1: Have a specialized chat for user profile creation. Where the user gets a template of authentication and registration profiles. The chatbot should check that information between fields is logically or semantically connected and that fields do not contradict each other. The chatbot after completing a form, responds with a form in JSON format for further website functions.

Sub-System Goal 2: Have a specialized chat for volunteer onboarding, after creating a user profile or after user authentication. Where additional user fields such as competencies or additional information such as gender, age, etc. It will allow to get more personalized experience based on provided user interest and characteristics. Also, it will possibly open more features as in career portals. It could allow users to find specific tasks and possibly subscribe to them.

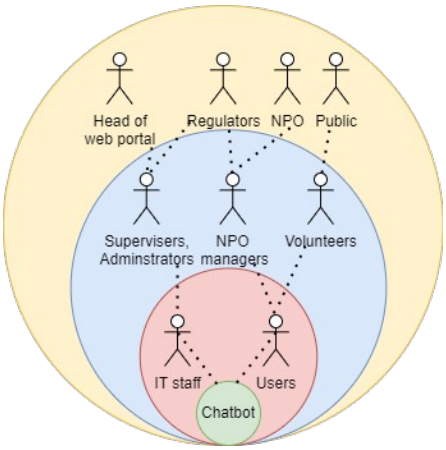
Sub-System Goal 3: Have a specialized chat for task creation. Where the user could get a template of a task based on his data. The chatbot should check that information between fields is logically or semantically connected and that fields do not contradict each other.

Requirements

Summary System Description

(Prototype for) AI-assisted chatbot for Austrian volunteers web portal.
The system based on modelfile configuration shall help to users by generating answers for users input. As An Ollama Modelfile is a configuration file that defines and manages models on the Ollama platform. Create new models or modify and adjust existing models through model files to cope with some special application scenarios.(GPU-mart)The system shall have authorization before accesing chatbot

Stakeholder Identification



Onion diagram

System:

The system represents the product or solution that is being delivered by the project (Olson 2013)

- Chatbot

Business System:

This layer represents the Business System and entails not just the final product, but those stakeholders who interact directly with it, like operators. (ConceptDraw)

- IT staff(web master, server technician etc)
- Users

Web application:

Those stakeholders who are functional “beneficiares” according Ian Alexander. These are the other stakeholders within the organization who may not interact directly with the solution who benefit from it. (Alexander 2003)

- Volunteers
- NPO managers
- Supervisers, Administrators

Public Environment:

Those stakeholders who are wider environment in which the organization operates. This layer is populated with stakeholders who are outside the firm but who are still important. As public or various pseudo-governmental organizations has a crucial stake in most projects. (Alexander 2003)

- Head of web-portal environment
- Non Profit Organization
- Public / Citizens
- Regulators
    - AI Act of European AI Office (AI Act 2024)
    - The Federal Ministry of Social Affairs, Health, Care and Consumer Protection
    - Municipal/ Local offices and representatives of European AI Office and Federal Ministry

The functional and nonfunctional requirements (using the EARS template)

List of functional requirements

Authentification

FR100: The system shall allow user create account (registrate) for the purpose of providing conversation for every user

FR101: The system shall allow user to his/her login (authentificate) by entering his/her password

FR102: The system shall allow user change password based on his/her account

FR103: The system shall use SQLite db as DBMS for easier integration with other parts of web portal

Chatbot

FR200: The LLM-powered chatbot shall respond to user inputs in natural language.

FR201: The chatbot should generate an answer for every message from user input.

FR202: The system shall be able to understand the user's intention from their input.

FR203: The bot shall extract specific information (entities) from the user's message, such as dates, names, or task information.

FR204: The bot shall have context management and conversation flow

> **SFR2040:** The system shall maintain the context of a conversation across multiple turns, remembering prior user inputs and responses.
>
> **SFR2041:** The system should handle ongoing conversations, where it can maintain state and carry information across multiple exchanges.
>
> **SFR2042:** The chatbot should continuously update its internal state to reflect new information from the conversation and respond in a relevant manner based on the current state.
>
> **SFR2043:** The system shall decide the next action (asking for more information, providing an answer, executing a task) based on the current state.

FR205: The system shall handle misunderstandings and have adaptability

> **SFR2050:** The system shall be able to handle misunderstandings, detect when it cannot understand, and ask clarifying questions.
>
> **SFR2051:** The chatbot should recognize when it cannot fully understand a user's input and ask clarifying questions to resolve ambiguities.
>
> **SFR2052:** The system shall adapt to individual users by learning preferences over time.

Add specific functions requirements ask user profile registration

List of nonfunctional requirements

Chatbot

NFR 200: The LLM-powered chatbot’s architecture shall allow for updates and bug fixes to be deployed without downtime.

NFR 201: The system should handle a large number of concurrent users and scale efficiently without performance degradation.

Actors and Agents

Primary roles (who use the system in their daily activities) in product system

#	Primary Actor	Type	Description
1	User of chatbot	Human	Responsible for 1. registration process via entering personal details in special chat or webpage 2. autentification process before starting chat for sub-system goals 2 and 3 3. entering messages and questions for defining exact topic of user interest
2	Chatbot Processing Model	AI	Provides core logic for answering on messages in natural language processing 1. Configured as chatbot assistant for Austrian volunteers portal 2. Identifying individual chat and conversation history 3. Fulfill specific goal of sub-system chats
3	IT staff	Human	Responsible for 1. Maintenance technical equipment such as server equipment 2. Problems and bug solving 3. Help users troubles and questions with interaction to chatbot instance

Prototyping Scenario

Have conversation as chatbot with predefined role and knowledge specialization

It contains Use case according template, Architecture Diagram and Screenshot examples according requirements template

Use Cases

ID	UC100-Auth-Login-page
Description	User logs into app
Actors	Volunteer/NPO manager as User of chatbot
Stakeholders	NPO managers, Volunteers
Pre-Conditions	The system has access to the internet via the device's WLAN access point
Success end condition:	The user is logged into the app and into the chatbot in the background
Failure end condition:	The user is not logged in and is being asked to contact support to verify credentials & permissions

Main Success Scenario

1	The user enters his/her user info (user id, password)
2	The user submits the user info
3	The systems checks whether the user is permissioned to use the chatbot app
4	The system creates a session for the user for submitting API requests LLM engine
5	The app launches and displays the main screen/menu

Alternative Scenarios

Exception Scenario

3.A1.1	Checking the user credentials results in an authentication error (unknown/invalid user)
3.A1.2	The system asks the user the verify credentials and to try to login again
3.A2.1	Checking the user credentials results in an authorisation error (user is known but not permissioned to use the app)
3.A2.2	The system informs that permission to use the app is missing and to contact support Before trying to log in again
4.A3.1	The system is unable to create a session with the chatbot system API (either authentication, authorisation or general technical error)
4.A3.2	The system asks the user to verify network connection and permissions Inside the web application system before trying to log in again

ID	UC101-chat-for-user-creation-

Description	Visitor uses chatbot to create profile
Actors	User of chatbot
Stakeholders	Public, NPO
Pre-Conditions	The system has access to the intranet via the device's WLAN access point and user has not logged
Success end condition:	The user created from chat conversation
Failure end condition:	The website did not created user from chat for user creation

Main Success Scenario

1	The chatbot checks whether the user is not logged
2	The user starts his conversation with chatbot
3	The user sends user details as name, username, last name, email and password
4	The chatbot checks the validity of fields
4	The system generates JSON file from user entries and volunteer role as default
5	The app saves correct json as entity in database
6	The app displays authorization page for log in

Alternative Scenarios

Exception Scenario

3.A1.1	Checking the user not logged
3.A1.2	The system returns to main page and show alert message that user alredy logged in
3.A2.1	Checking the user entries for relevance in task fields
3.A2.2	The system informs that entries does not correspond with fields as password and password confirmation. Chatbot ask user to resend data in appropriate format
4.A3.1	The system is unable to create a task entity with the chatbot database API as username or email is exists
4.A3.2	The system show system error in chat with appropriate field as it already exists Inside the chatpage system ask to try send other data substitute new data and initializes another user creation function

ID	UC102-chat-for-volunteer-profile
Description	User logs into app with volunteer profile
Actors	Volunteer as User of chatbot
Stakeholders	Voluunteer, Public
Pre-Conditions	The system has access to the intranet via the device's WLAN access point and user has logged as volunteer
Success end condition:	The user completed volunteer profile from chat conversation
Failure end condition:	The website did not completed volunteer profile with data from chat

Main Success Scenario

1	The chatbot checks whether the user logged as volunteer
2	The user starts his conversation with chatbot
3	The user sends voluunteer details as description, schedule, gender, competences etc.
4	The chatbot checks the validity of fields
4	The system generates JSON file from user entries.
5	The app saves correct json as entity in database
6	The app displays user page

Alternative Scenarios

Exception Scenario

3.A1.1	Checking the user not logged as volunteer
3.A1.2	The system returns to main page and show alert message that access is denied as user profile is not Volunteer
3.A2.1	Checking the user entries for relevance in task fields
3.A2.2	The system informs that entries does not correspond with fields as competences are out of list of competences. Chatbot ask user to resend data in appropriate format
4.A3.1	The system is unable to create a task entity with the chatbot database API as data formats is wrong
4.A3.2	The system show system error in chat with appropriate field which syntaxis is inappropriate Inside the chatpage system ask to try send json initialization request again to chatbot

ID	UC103-chat-for-task-creation-
Description	User logs into app with NPO manager profile
Actors	NPO manager as User of chatbot
Stakeholders	NPO managers, NPO
Pre-Conditions	The system has access to the intranet via the device's WLAN access point and user has NPO manager profile
Success end condition:	The NPO managers task created from chat conversation
Failure end condition:	The website did not created chat for NPO manager

Main Success Scenario

1	The chatbot checks whether the user is permissioned to create tasks
2	The user starts his conversation with chatbot
3	The user sends task details as name, description, startdate, enddate
4	The chatbot checks the validity of fields
4	The system generates JSON file from user entries.
5	The app saves correct json as entity in database
6	The app displays page with users tasks

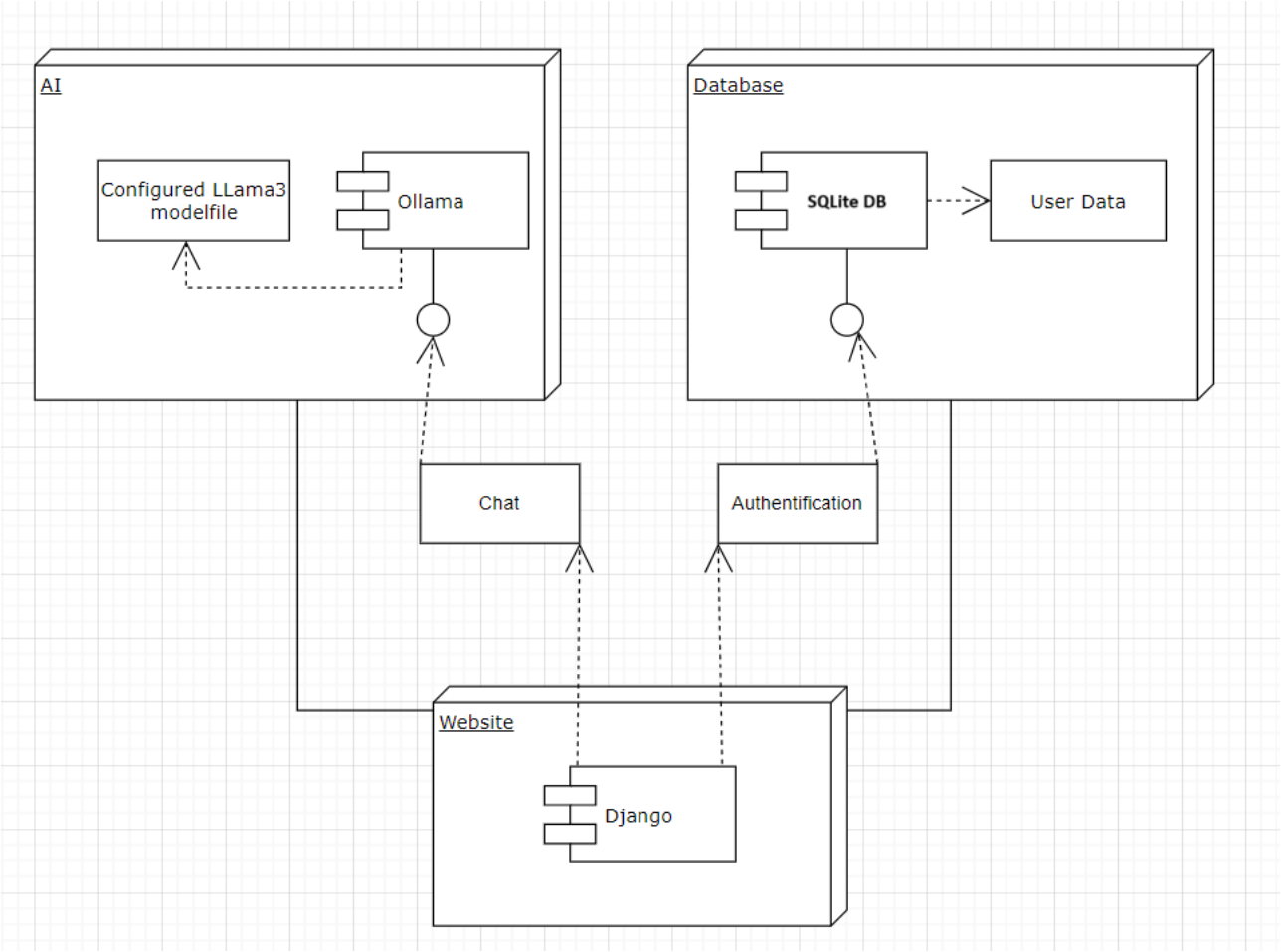
Alternative Scenarios

Exception Scenario

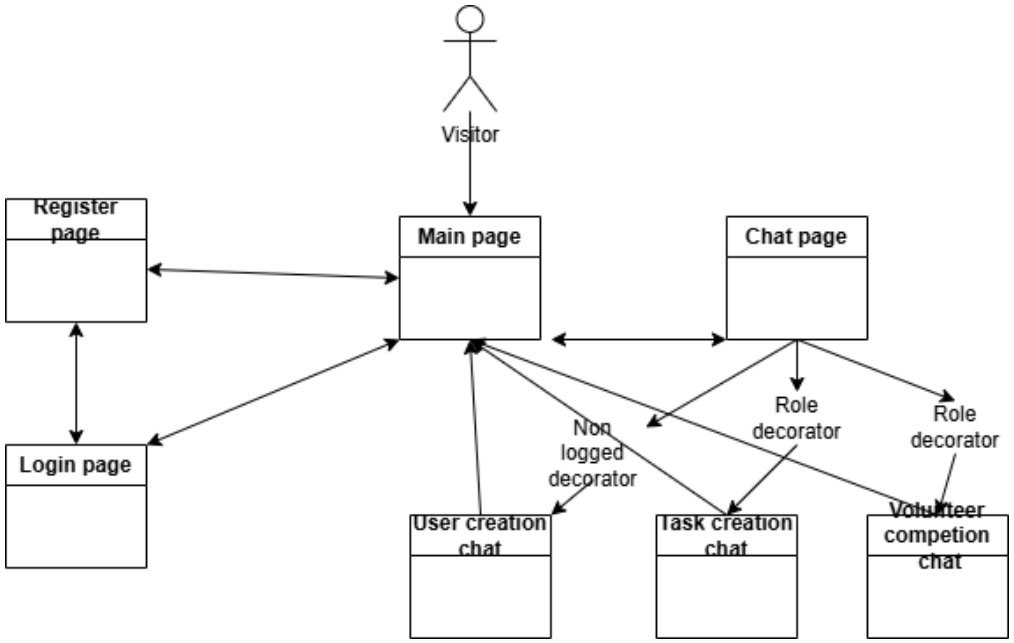
3.A1.1	Checking the user permission(not NPO manager profile)
3.A1.2	The system returns to main page and show alert message that access is denied as profile is not NPO manager
3.A2.1	Checking the user entries for relevance in task fields
3.A2.2	The system informs that entries does not correspond with fields as dates exepts system restrictions. Chatbot ask user to resend data in appropriate format
4.A3.1	The system is unable to create a task entity with the chatbot database API
4.A3.2	The system asks the user to verify network connection and permissions Inside the chatpage system ask to try send json initialization request again to chatbot

Concept

Base architecture diagram



Web model



Data model

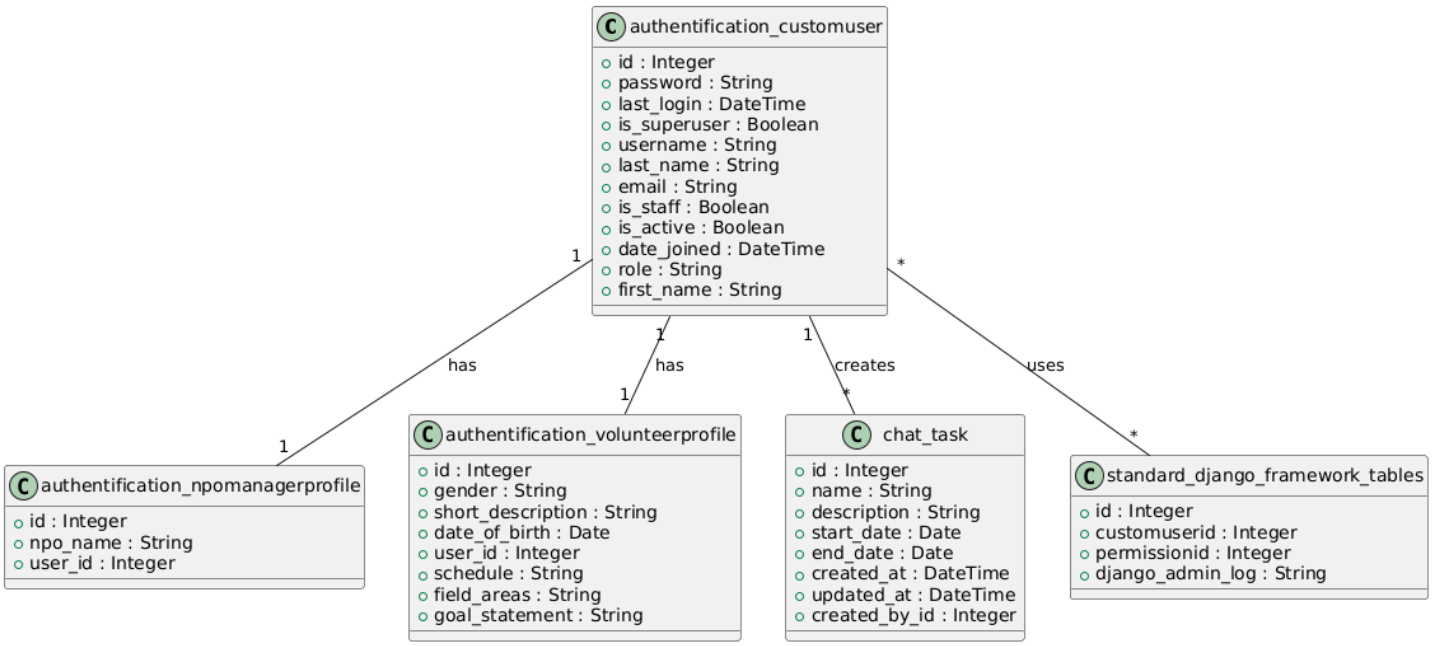
In this section I have the explanation of base models which are used in web application.

I have the User model for all users. It means that volunteers, NPO managers and admin have all general fields. All role specific access, rights and fields are stored in specific profiles.

- Centralized User Management: By using one base User model, the core authentication (e.g., login, password, permissions) remains centralized and consistent across the application.
- Specialized Profiles for Custom Data: The user model could be extended to hold domain-specific fields (e.g., user roles, additional contact details) without cluttering the base User model itself. This keeps the responsibility of the base User model limited to authentication and core properties.
- Dynamic Role Management: Specialized profile models for different types of users (e.g., Volunteer, NPO Manager). These models can hold different sets of data based on their specific needs or business logic, such as varying fields or settings.
- Efficient Rights and Access Control: Permissions and access rights can be customized at the profile level. A common practice is to define different permissions or access rules for each user role, which can be easily handled by specialized profile models.

This approach have several benefits:

- Easier to Scale User Models: As application grows to introduce more features (e.g., new user roles or additional fields specific to user types), Users can be extended the profile model without modifying the base User model. This makes it easier to scale app and avoid repeated changes to the user-related codebase.
- Better Data Organization: Having separate profiles for different roles makes the database schema more modular, and allows to avoid overloading the base user table with unnecessary fields.
- Data Integrity and Performance: By splitting data into logical profiles for different roles, it improve the integrity and performance of database queries. When accessing a specific profile, app don’t have to retrieve unnecessary information for other types of users.



UML diagram generated with Plantext UML Editor

User (CustomUser in code) contains:

- Role (volunteer or NPO manger)
- Username (unique for each user)
- Email (unique for each user)
- First name
- Last name (Surname)
- Password

Also database store some internal fields as Id, last_login, is_superuser, is_staff, is_active, date_joined

Volunteer profile which extends user contains:

- User id as foreign key to user
- Gender (Male, Female, Other)
- Short description
- Date of birth
- Competencies, which contains predefined list of competencies and values from 0 to 3 (field_areas)
- Interests
- Schedule (preferred available time)
- Goal statement

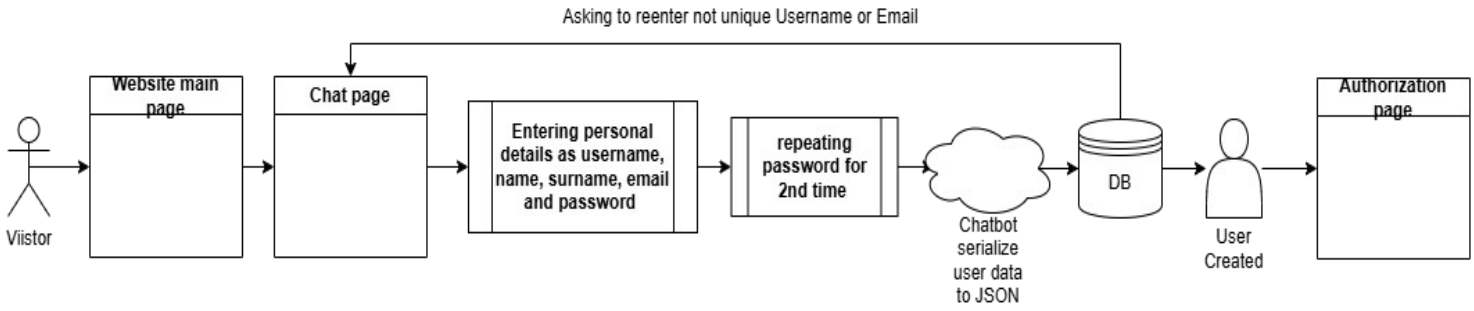
NPO manager contains:

- User id as foreign key to user
- NPO name

Then Task model contains:

- Name
- Description
- Start date
- End date
- Created by which is foreign key to user
- Date of creation (created_at)
- Date of update (updated_at)

The table standard_django_framework_tables is schematic representation of User class relations with standard Django framework generated tables



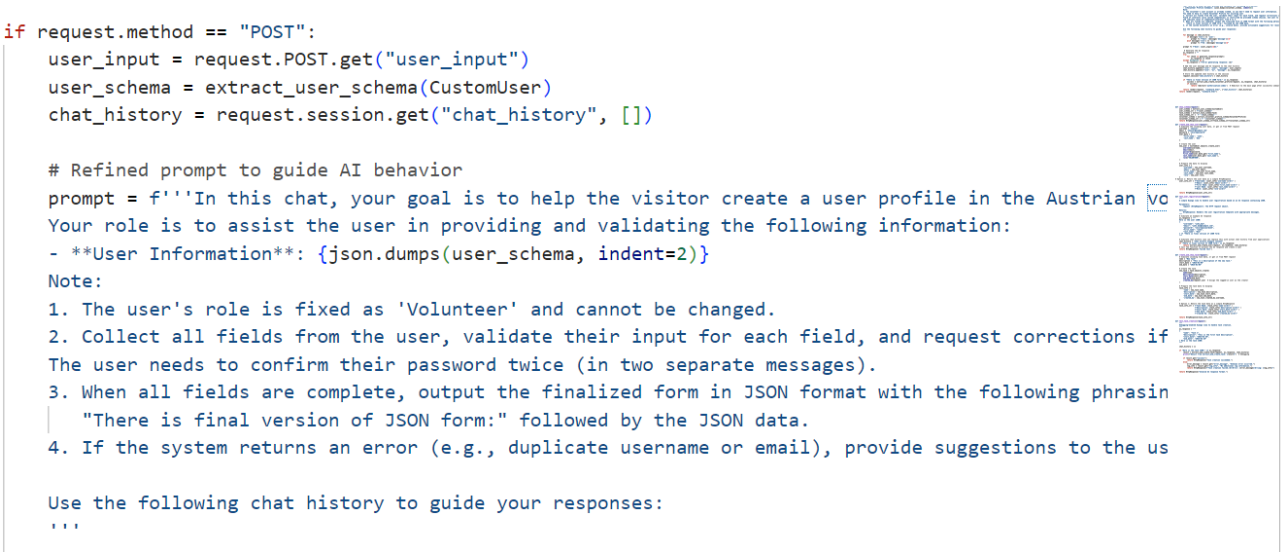
Pipeline of user creation chat

Before access the chat page for user creation Django check with @unauthenticated_user decorator that user are not logged in.

```python
def role_required(required_role):
    def decorator(view_func):
        @wraps(view_func)
        def _wrapped_view(request, *args, **kwargs):
            if not request.user.is_authenticated:
                messages.warning(request, "You need to log in to access this page.")
                return redirect('authentification:signin')  # Redirect to login page
            if request.user.role != required_role:
                messages.error(request, "Access denied. You do not have permission to access this page.")
                return redirect('authentification:index')  # Redirect to main page
            return view_func(request, *args, **kwargs)
        return _wrapped_view
    return decorator

def unauthenticated_user(view_func):
    def wrapper(request, *args, **kwargs):
        if request.user.is_authenticated:
            # Redirect authenticated users to a main page
            messages.error(request, "Access denied. You already have user account.")
            return redirect('authentification:index')  # Redirect to main page
        return view_func(request, *args, **kwargs)

    return wrapper
```

Volunteer profile and task creation check for user logged in and specific role, volunteer and npo manager respectively with special decorator function.

The chat conversations of specific page follows main goal of the chat which is aimed in initial prompt for the ollama bot. In case of usercreation page it has such prompt.



```python
if request.method == "POST":
    user_input = request.POST.get("user_input")
    user_schema = extract_user_schema(CustomUser)
    chat_history = request.session.get("chat_history", [])

    # Refined prompt to guide AI behavior
    prompt = f'''In this chat, your goal is to help the visitor create a user profile in the Austrian vo
    Your role is to assist the user in providing and validating the following information:
    - **User Information**: {json.dumps(user_schema, indent=2)}
    Note:
    1. The user's role is fixed as 'Volunteer' and cannot be changed.
    2. Collect all fields from the user, validate their input for each field, and request corrections if
    The user needs to confirm their password twice (in two separate messages).
    3. When all fields are complete, output the finalized form in JSON format with the following phrasin
       "There is final version of JSON form:" followed by the JSON data.
    4. If the system returns an error (e.g., duplicate username or email), provide suggestions to the us

    Use the following chat history to guide your responses:
    '''
```

Each prompt contains main goal for chat boat which it need to complete during conversation and main features for what bot need to pay attention as fixed role of the user. Also in prompt I formulate condition, which later I use to activate special listener and try to save JSON file as database entity

For every prompt in the app the schema of the class is generated dynamically with specific function. This approach allows to change class model and chatbot functionality independently without hardcoding and rules as in rule-based chatbots. For example volunter profile dynamic schema

```python
def extract_volunteer_profile_schema(model):
    """
    Dynamically extract the schema of the VolunteerProfile model with field names and descriptions,
    ensuring fields are included in a specific order and their help_text or default descriptions are used
    """
    excluded_fields = ["id", "user"]  # Exclude fields like ID and user relationship

    # Predefined order of fields for better readability
    field_order = [
        "gender",
        "short_description",
        "date_of_birth",
        "interests",
        "goal_statement",
        "competencies_areas",
        "schedule"
    ]

    schema = {}
    for field in model._meta.get_fields():
        if field.concrete and not field.is_relation and field.name not in excluded_fields:
            # Add help_text or a default description if help_text is not available
            schema[field.name] = force_str(field.help_text or " ")

    # Handle special fields like interests, competencies_areas, and schedule
    schema["interests"] = "Tags representing areas of interest, e.g., 'Technology', 'Health'."
    schema["competencies_areas"] = (
        "A dictionary mapping FIELD_CHOICES to levels (0-3), e.g., "
        "{'ENVIRONMENT': 2, 'EDUCATION': 3}. "
        f"Available FIELD_CHOICES: {dict(model.FIELD_CHOICES)}."
```

```python
    # Handle special fields like interests, competencies_areas, and schedule
    schema["interests"] = "Tags representing areas of interest, e.g., 'Technology', 'Health'."
    schema["competencies_areas"] = (
        "A dictionary mapping FIELD_CHOICES to levels (0-3), e.g., "
        "{'ENVIRONMENT': 2, 'EDUCATION': 3}. "
        f"Available FIELD_CHOICES: {dict(model.FIELD_CHOICES)}."
    )
    schema["schedule"] = (
        "A dictionary mapping days of the week to available time slots, e.g., "
        "{'Monday': ['14:00-16:00'], 'Friday': ['10:00-12:00', '15:00-17:00']}. "
        f"Valid days: {[day[0] for day in model.DAYS_OF_WEEK]}."
    )

    # Reorder the schema based on the predefined order
    ordered_schema = {field: schema.get(field, " ") for field in field_order}

    return ordered_schema
```

As for this goal I have complex class attributes as schedule, which I reflect in code as dictionary and string in database. I additionally add help texts for fields for easier understanding for bot. After that bot understands what are expected from user and in every users message information adjusted to fill every field. So all prompt design use quite similar prompt architecture as Guangzhi Sun. In his team work they also created prompt with general task definition (in my case goal definition), in-context example (explains how LLM should act in specific case, in my situation I show the skeleton to fill with specific values and explanations), LKI (which was possible values for some slot, in my problem some values which could not be taken as end date earlier start date or default volunteer role value for user creation) and query (specific situation which is evaluating, in app this is users inputs) (Guangzhi Sun)

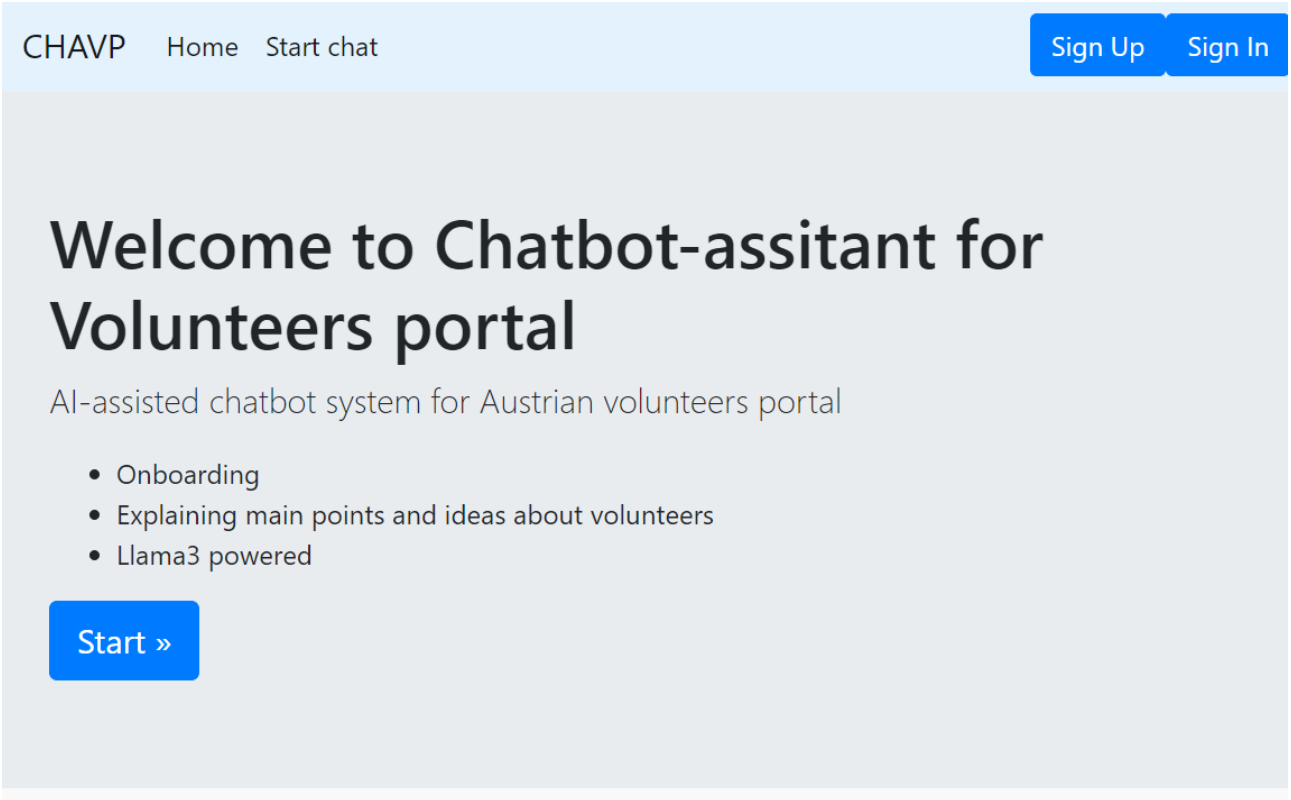
Chatbot awaits that user fills all required fields filled the calls trigger to try save entity.

```python
# Add the user message and AI response to the chat history
chat_history.append({"role": "user", "message": user_input})
chat_history.append({"role": "ai", "message": ai_response})

# Store the updated chat history in the session
request.session["chat_history"] = chat_history

if "There is final version of JSON form:" in ai_response:
    result = extract_and_create_task(request, ai_response, chat_history)
    if result.get("success"):
        return redirect('chat:user_tasks', username=request.user.username)
```

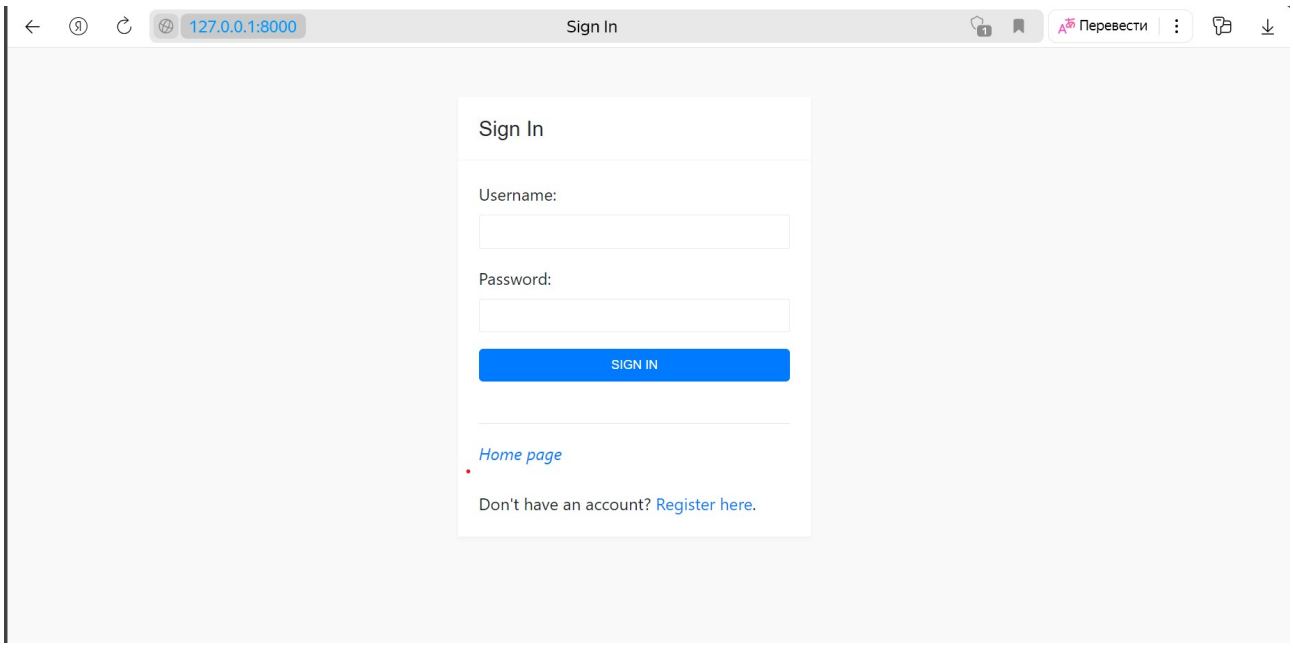
extract_and_create_task function additionally validate data for field and application restrictions as ensuring that every attribute has appropriate value and then try to create task. If everything is correct it will write log message and redirect to user's tasks page.

```python
def extract_and_create_task(request, ai_response, chat_history):
    json_pattern = r'\{.*?\}'
    json_fragments = re.findall(json_pattern, ai_response, re.DOTALL)

    print(f"Extracted JSON fragments: {json_fragments}")  # Debugging

    for fragment in json_fragments:
        try:
            task_data = json.loads(fragment.strip())
            print(f"Parsed JSON: {task_data}")  # Debugging
            name = task_data.get("name")
            description = task_data.get("description")
            start_date = task_data.get("start_date")
            end_date = task_data.get("end_date")
            # Validation logic...
            if not name or not description or not start_date or not end_date:
                print("Validation failed. Missing required fields.")
                break

            # Assume Task.objects.create() works as expected for now
            #print(f"Task created: {task_data}")
            new_task = Task.objects.create(
                name= name,
                description=description,
                start_date= start_date,
                end_date= end_date,
                created_by=request.user,  # Set the current logged-in user as the creator
            )
            return {"success": True, "log_info": "Task created successfully."}
```

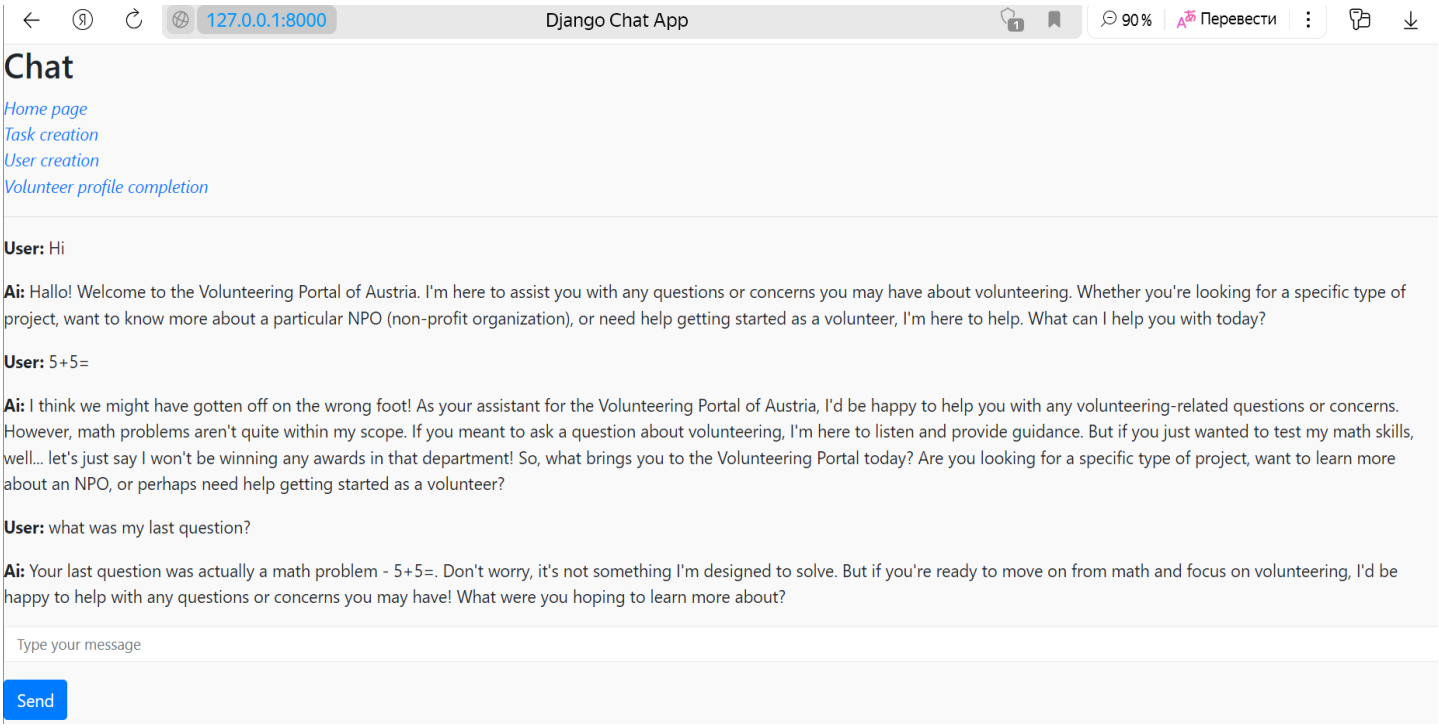
User Scenarios



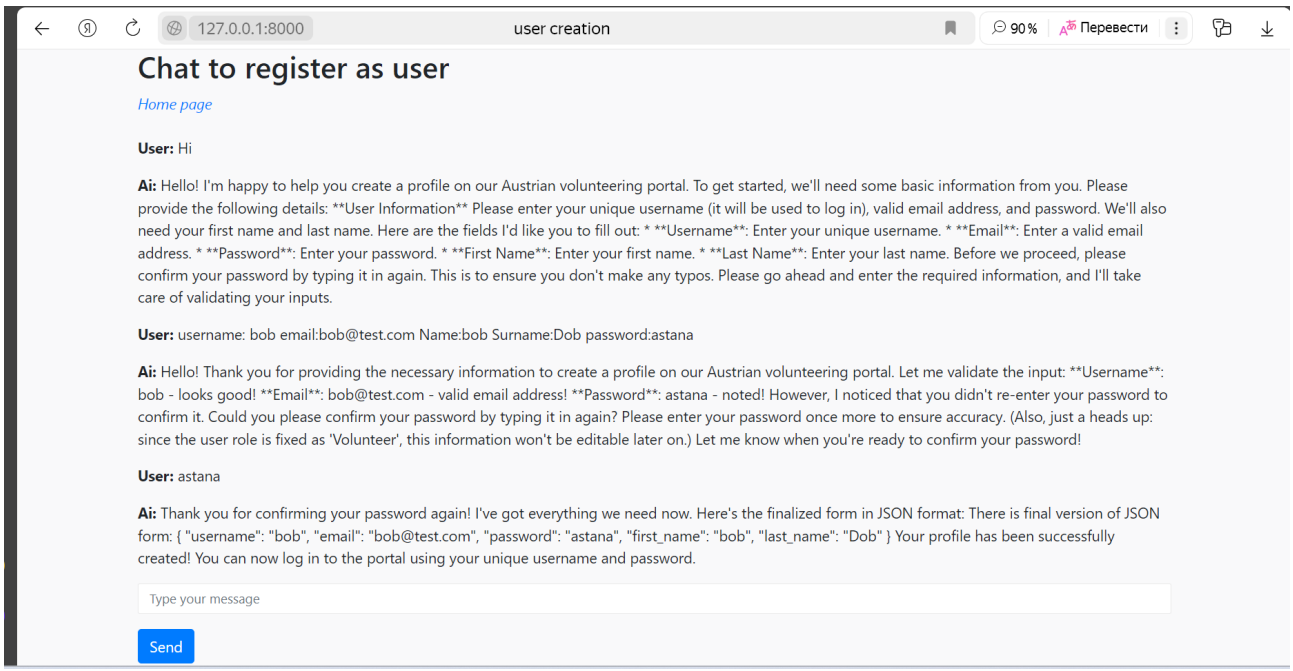
Application main page where user can start chat by buttons or log in or register via traditional web pages



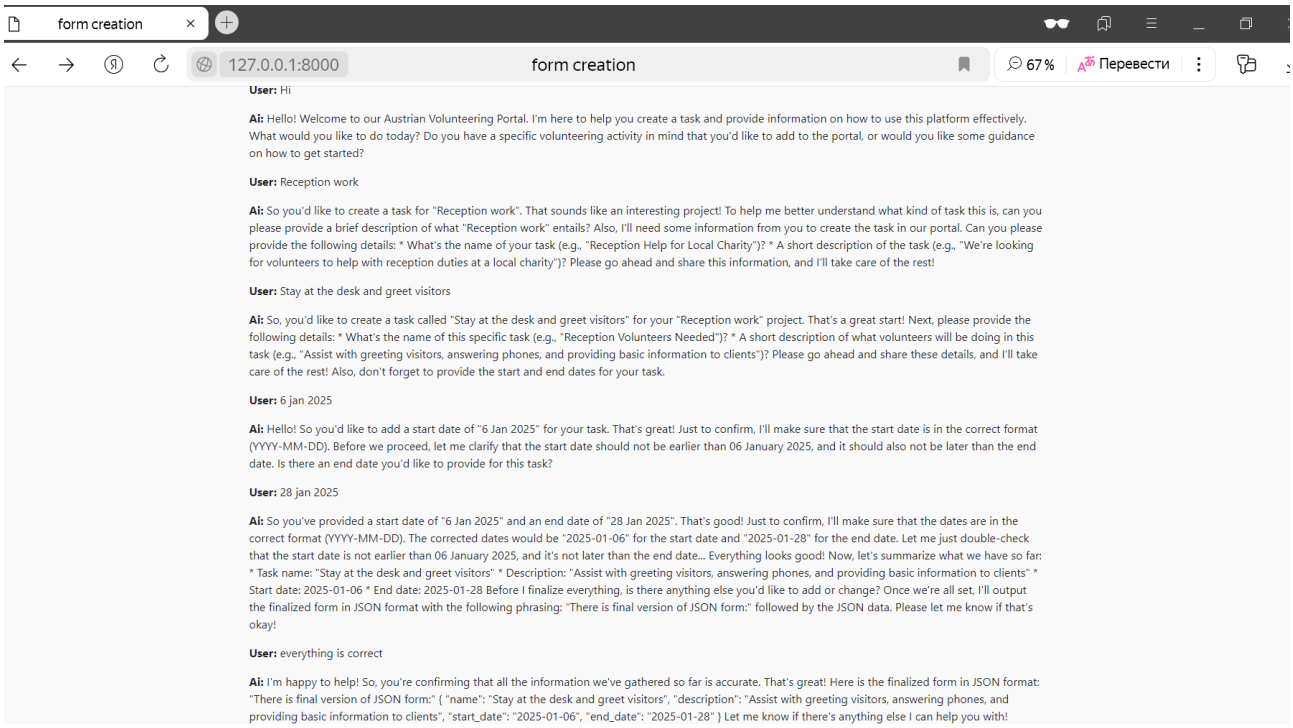
On the login page, user can enter username and password. Afterward, user can easily access other chats on another screen. For individuals without an existing account, there is an option to create a new one.



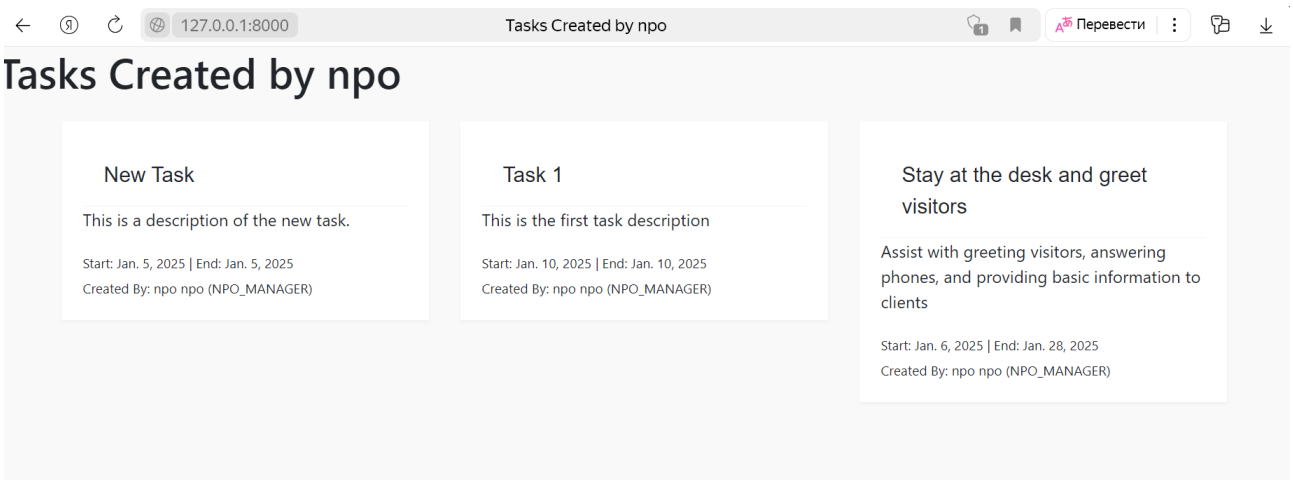
Main chat page as chatgpt. In this page bot will try to talk about Austrian Volunteering portal and avoid other themes. As it has chat history on the page it can support sequential dialog. In this page are links to all sub-system goals chats.



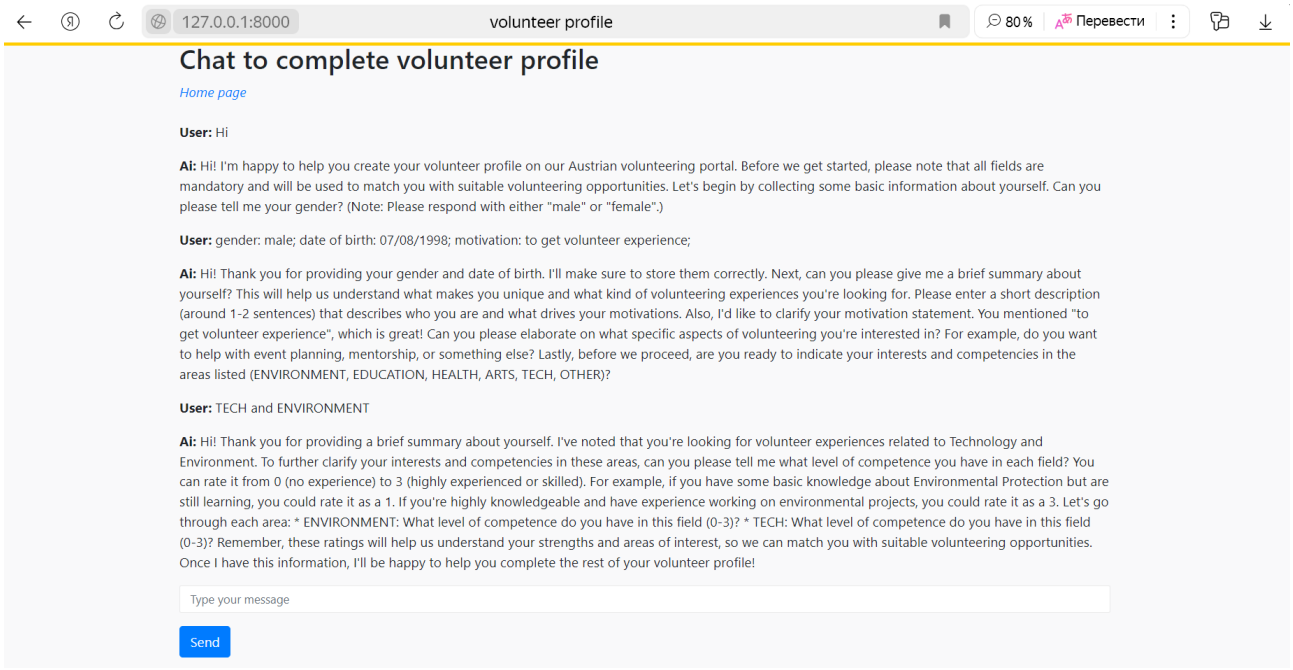
User creation chat. Here visitor sends all information to create profile. If everything correct chatbot will create final version of json and will trigger user creation function.



task creation chat example. In this chat bot transforms date to standard format of class attribute and checks that start and end date did not cause contradiction. After that it ask confirmation and create final JSON for task entity



After task creation new task will appear in user created tasks page where all task stored



During completing the volunteering profile LLM bot use various questions to formulate and gather information. Sometimes it adjust complex fields to several messages for careful analysis.

References

1. AI Office of the European Commission, “The AI Office: What is it, and how does it work?” https://artificialintelligenceact.eu/the-ai-office-summary/, − published 21 March 2024

2. Weizenbaum, J. (1966). "ELIZA - A Computer Program for the Study of Natural Language Communication Between Man and Machine." *Communications of the ACM*, 9(1), 36-45.

3. Tiffany, ChatInsight “A Comprehensive Guide to Build a Rule-Based Chatbot” https://www.chatinsight.ai/chatbots/rule-based-chatbot/ , - published 20 February 2024 –

4. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... Amodei, D. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165.

5. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. arXiv preprint arXiv:1706.03762

6. Llama Team, AI @ Meta. The Llama 3 Herd of Models. https://scontent-vie1-1.xx.fbcdn.net/v/t39.2365-6/463020162_522238820565582_8192401983671993921_n.pdf?_nc_cat=108&ccb=1-7&_nc_sid=3c67a6&_nc_ohc=ZGqgxjTvDMYQ7kNvgEk4Tt9&_nc_zt=14&_nc_ht=scontent-vie1-1.xx&_nc_gid=AJtJhR8dBRBkApUnavDR2xw&oh=00_AYBLgff_1hfIRxucPP9FKolsU3_AzlpL_ffuA-bAmcWeYw&oe=67270659. published July 23, 2024.

7. GPU-mart, How to Customize LLM Models with Ollama's Modelfile https://www.gpu-mart.com/blog/custom-llm-models-with-ollama-modelfile

8. Paolo Confino, Fortune https://fortune.com/2024/07/26/jpmorgan-chatbot-llm-suite-chatgpt-ai-machine-learning-wall-street-banking-finance/ published 26 July 2024

9. David Olson, Bawiki “Stakeholder Onion Diagram What is it?” http://www.bawiki.com/wiki/Stakeholder-Onion-Diagram.html published 2013

10. ConceptDraw “Stakeholder Onion Diagrams” https://www.conceptdraw.com/examples/management-stakeholder-onion-diagrams

11. Ian Alexander “Stakeholders – Who is Your System For?” https://www.scenarioplus.org.uk/papers/stakeholders/stakeholders.htm in Computing & Control Engineering, Vol 14, Issue 1, pp 22-26, April 2003

12. Guangzhi Sun et al. "Speech based Slot filling unsing Lanrge Language models" In: (2023) arXiv: 2311.07418