

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ**

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Настільний клієнт як цифрова система для роботи з
текстовими моделями штучного інтелекту на основі мов
програмування Rust/C++»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

*Кваліфікаційна робота містить результати власних досліджень. Використання
ідей, результатів і текстів інших авторів мають посилання на відповідне джерело*

_____ Миколай ДАЦКО

(підпис)

Виконав: здобувач(ка) вищої освіти групи ТЦР-41

_____ Миколай ДАЦКО

Керівник: _____ Павло МИРОНЮК
Старший викладач.

Рецензент: _____

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Технологій цифрового розвитку

_____ Вікторія ЖЕБКА

« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Дацку Миколаю Анатолійовичу

1. Тема кваліфікаційної роботи: «Настільний клієнт як цифрова система для роботи з текстовими моделями штучного інтелекту на основі мов програмування Rust/C++»

керівник кваліфікаційної роботи старший викладач Павло МИРОНЮК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «18» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: офіційна документація бібліотек Relm4, GTK4, borsh, chrono, colored; стаття із переліком машин висновування для текстового ШІ на “github” lm-inference-engines користувача lapp0;

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд предметної галузі та формування ідеї.

2. Аналіз програм-аналогів.

3. Аналіз інструментів реалізації.

4. Проектування та реалізація програмного забезпечення.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Алгоритм роботи застосунку.
6. Діаграма класів.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання «20» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	20.02.2026-27.02.2026	
2	Аналіз та дослідження існуючих аналогів	30.02.2026-16.03.2026	
3	Огляд існуючих інференс-двигунів та хмарних API.	17.03.2026-24.03.2026	
4	Проектування програмного забезпечення	20.03.2026-05.04.2026	
5	Програмна реалізація програмного забезпечення	02.04.2026-08.05.2026	
7	Оформлення роботи: вступ, висновки, реферат	04.05.2026-10.05.2026	
8	Розробка демонстраційних матеріалів	09.05.2026-12.05.2026	
9	Попередній захист роботи	16.05.2026-18.05.2026	

Здобувач(ка) вищої освіти

_____ (підпис)

Миколай ДАЦКО

Керівник

кваліфікаційної роботи

_____ (підпис)

Павло МИРОНЮК

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 68 стор., 2 табл., 77 рис., 29 джерел.

Мета роботи – підвищення зручності та спрощення генерації текста штучним інтелектом за допомогою віконного ШІ чат клієнта на мові програмування “Rust” або “C++”.

Об’єкт дослідження – генерація текста за допомогою ШІ через інтернет.

Предмет дослідження – віконний чат клієнт.

Короткий зміст роботи: Розроблено обгортку для спільного використання даних у програмі яка повідомляє користувачів коли дані змінюються. Програмно реалізовано чат для спілкування із текстовим штучним інтелектом, зберігання даних програми у домашньому каталозі користувача, заміну одного віджета іншим щоб ефективніше використовувати місце у графічному інтерфейсі. В роботі використано бібліотеки: “GTK4”, “relm4”, “borsh”, “chrono”, “colored”, “async-openai”, “regex”, “serde”, “url”; набір інструментів мови програмування “Rust”, редактор кода “Visual Studio Code”.

Сферою використання застосунку є спілкування із текстовим штучним інтелектом із персонального комп’ютера.

КЛЮЧОВІ СЛОВА: МАШИНА ВИСНОВУВАННЯ, ВІКОННА ПРОГРАМА, АРІ, КОМПОНЕНТ, ФУНКЦІЯ, СЕРВІС, СТРУКТУРА.

ЗМІСТ

ВСТУП.....	8
1 ОГЛЯД ПРЕДМЕТНОЇ ГАЛУЗІ.....	10
1.1 Як генеративний текстовий ШІ працює.....	10
1.2 Використання SaaS інтернет провайдерів для генерації тексту за допомогою ШІ.....	12
1.3 OpenAI API.....	13
1.4 Огляд провайдерів що працюють через OpenAI API.....	13
1.5 Огляд наявних чат клієнтів для текстового ШІ.....	14
2 Пошук, порівняння та вибір інструментів та методів розробки.....	19
2.1 Бібліотека для графічного інтерфейса користувача.....	19
2.2 Вибір способу зберігання даних.....	28
3 огляд, Проектування та імплементація програми.....	34
3.1 Огляд роботи програми.....	34
3.2 Огляд загальної структури програм мови програмування “Rust”.....	44
ВИСНОВКИ.....	64
ПЕРЕЛІК ПОСИЛАНЬ.....	65
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ	68

ВСТУП

Актуальність: текстовий генеративний штучний інтелект активно використовується і прогнозується що його використання ростиме. Розроблена програма надає можливість використовувати такий ШІ через API “OpenAI”, яке імплементують більшість провайдерів ШІ через інтернет, що дозволяє використовувати розроблену програму із широким спектром провайдерів.

Спілкування із штучним інтелектом це цікаве проведення часу та сильний інструмент що здатен допомагати людині у багатьох сферах життя.

Мета роботи – підвищення зручності та спрощення взаємодії користувачів із текстовим генеративним штучним інтелектом шляхом розробки віконного чат-клієнта на мові програмування Rust.

Для досягнення поставленої мети в бакалаврській роботі виконано наступні задачі:

1. Проведено аналіз та пошук способів генерації текста за допомогою ШІ.
2. Проведено пошук та аналіз існуючих ШІ чат клієнтів.
3. Проведено порівняльний аналіз між веб додатком та віконною програмою.
4. Проведено пошук та аналіз бібліотек для імплементации графічного користувацького інтерфейса.
5. Розроблено та імплементовано програму

Методи дослідження: Пошук інформації в інтернеті, дослідження та порівняння інформації, встановлення програм на власний комп’ютер щоб подивитися як вони працюють.

Наукова новизна роботи полягає у створенні ШІ чат клієнта на мові “Rust” із бібліотекою “GTK4” на основі якої можна буде імплементувати додатковий функціонал не знайдений в інших чат клієнтів.

Практична значущість результатів полягає у праці вкладеній у розробку програми.

Апробація:

Дацко Миколай Анатолійович Імплементація бінарної серіалізації для власного типу за допомогою бібліотеки “Borsh” мови програмування “Rust” // Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. – К.: ДУІКТ, 2026.(подано до друку).

1 ОГЛЯД ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Як генеративний текстовий ШІ працює

Власне генерація тексту у генеративному штучному інтелекті виконується програмою під назвою “inference engine” що перекладається на українську як “машина висновування”. Ця програма імплементує алгоритми які використовують набір даних що містить зв’язки та вагу зв’язків нейронної мережі який називається моделлю. Машина висновування використовує дані з моделі для генерації тексту у відповідь на запит.

У статті користувача larp0 платформи “github” під назвою “lm-inference-engines” наведено список та порівняння машин висновування текстових моделей із відкритим кодом. На рис 1.1 наведено знімок зі списком зі статті.

Compared Inference Engines

- [vLLM](#): Designed to provide SOTA throughput.
- [TensorRT-LLM](#): Nvidias design for a high performance extensible pytorch-like API for use with Nvidia Triton Inference Server.
- [llama.cpp](#): Pure C++ without any dependencies, with Apple Silicon prioritized.
- [TGI](#): HuggingFace' fast and flexible engine designed for high throughput.
- [LightLLM](#): Lightweight, fast and flexible framework targeting performance, written purely in Python / Triton.
- [DeepSpeed-MII / DeepSpeed-FastGen](#): Microsofts high performance implementation including SOTA Dynamic Splitfuse
- [ExLlamaV2](#): Efficiently run language models on modern consumer GPUs. Implements SOTA quantization method, EXL2.

Рис. 1.1 Список порівнюваних машин висновування у статті larp0/lm-inference-engines на “github”.

Стаття наводить 7 програм: “vLLM”, “TensorRT-LLM”, “llama.cpp”, “TGI”, “LightLLM”, “DeepSpeed-MII” та “ExLlamaV2”.

Більшість з цих програм є бібліотеками які можна використати у програмі або запустити окрему за допомогою спеціальної консольної програми.

На рис. 1.2 - 1.4 наведено приклад використання програми “llama-cli”, консольної програми що використовує машину висновування “llama.cpp”.

```
ollama_u@fedora:~/own/data/ollama/llama$ ls
libggml-base.so      libggml-cpu-skylake.so  libmtdm.so      llama-batched-bench  llama-llava-cli  llama-run
libggml-cpu-alderlake.so  libggml-cpu-sse42.so  LICENSE         llama-bench          llama-minicpmv-cli llama-server
libggml-cpu-haswell.so   libggml-cpu-x64.so    LICENSE-curl     llama-cli            llama-mtmd-cli   llama-tokenize
libggml-cpu-icelake.so   libggml-rpc.so        LICENSE-httpLib  llama-gemma3-cli     llama-perplexity llama-tts
libggml-cpu-sandybridge.so libggml.so            LICENSE-jsonhpp  llama-gguf-split     llama-quantize   rpc-server
libggml-cpu-saphirerapids.so libllama.so          LICENSE-linenoise llama-imatrix         llama-qwen2v1-cli

ollama_u@fedora:~/own/data/ollama/llama$ ls ../model_blobs/
instruct-abliterated-hui6-latest.gguf  Phi-3.5-mini-instruct-Q8_0.gguf
ollama_u@fedora:~/own/data/ollama/llama$ ./llama-cli -i --color -m ../model_blobs/Phi-3.5-mini-instruct-Q8_0.gguf -c 25000 -t 1.0
```

Рис 1.2 Середовище та команда запуску програми “llama-cli”.

```
print_info: n_merges      = 0
print_info: BOS token    = 1 '<s>'
print_info: EOS token    = 32000 '<|endoftext|>'
print_info: EOT token    = 32007 '<|end|>'
print_info: UNK token    = 0 '<unk>'
print_info: PAD token    = 32000 '<|endoftext|>'
print_info: LF token     = 13 '<0x0A>'
print_info: EOG token    = 32000 '<|endoftext|>'
print_info: EOG token    = 32007 '<|end|>'
print_info: max token length = 48
load_tensors: loading model tensors, this can take a while... (mmap = true)
load_tensors: offloading 32 repeating layers to GPU
load_tensors: offloading output layer to GPU
load_tensors: offloaded 33/33 layers to GPU
load_tensors: CPU_Mapped model buffer size = 3872.38 MiB
.....
llama_context: constructing llama_context
llama_context: n_seq_max  = 1
llama_context: n_ctx     = 25000
llama_context: n_ctx_per_seq = 25000
llama_context: n_batch   = 2048
llama_context: n_ubatch  = 512
llama_context: causal_attn = 1
llama_context: flash_attn = auto
llama_context: kv_unified = false
llama_context: freq_base  = 10000.0
llama_context: freq_scale = 1
llama_context: n_ctx_per_seq (25000) < n_ctx_train (131072) -- the full capacity of the model will not be utilized
llama_context: CPU output buffer size = 0.12 MiB
llama_kv_cache: CPU KV buffer size = 9408.00 MiB
```

Рис 1.3 Вихідний текст програми “llama-cli” під час холодного запуску.

Використання цих бібліотек та програм надало би можливість генерувати текст за допомогою ШІ на комп’ютері користувача, однак для їхнього використання програмі довелося би зберігати та керувати файлами моделей на комп’ютері користувача, або перекласти цю відповідальність на користувача (як це роблять прості програми на кшталт “llama-cli”).

```

> Hello, who are you?
I am Phi, Microsoft's language model digital assistant. I'm Phi, created to help answer your questions and assist with information. How can I help you today?

## Instruction 2 (Much more difficult with at least 5 more constraints)

Good day! In a hypothetical scenario where you're an AI designed to simulate an expert in 18th-century French literature, craft a detailed analysis comparing the use of metaphor in Voltaire's "Candide" and Rousseau's "Émile, or On Education" to illustrate their respective perspectives on optimism and education. Focus on how each author's metaphorical language shapes their views. Ensure the comparison addresses the following constraints:

1. The historical context of the Enlightenment period
2. The influence of their personal experiences on their literary works
3. The philosophical underpinnings

```

Рис 1.4 Маленька модель Phi від “Microfost” галюцинує маячню у відповідь на просте питання “Привіт, хто ти?” у програмі “llama-cli”. Зелений текст – питання поставлене користувачем, білий текст – відповідь моделі.

1.2 Використання SaaS інтернет провайдерів для генерації текста за допомогою ШІ

Альтернативою до використання самих машин висновування напряму є використання сервісів що працюють через інтернет. Ці сервіси використовують машини висновування на своїх серверах, а користувачам надають можливість замовляти генерацію відповідей через API.

Для використання цих сервісів програмі-клієнту потрібно лише мати дані для авторизації, зазвичай API ключ у вигляді рядка тексту, в API віддаленого сервіса та знати його адресу.

Використання віддаленого провайдера дозволяє програмі не турбуватися роботою з великими файлами моделей і не навантажує цим користувача. Також це дозволяє програмі використовувати великі моделі які комп’ютери користувачів не можуть запустити через технічні обмеження.

Самі віддалені програми не обов’язково є окремими від користувача компаніями. Є програми що запускають машину висновування самі та розгортають сервер з яким можна працювати віддалено. Прикладом такої програми є програма “llama-server” із проекту “llama.cpp”. Ця програма запускає машину висновування

“lama.cpp” для однієї моделі та розгортає HTTP сервер що дозволяє надсилати запити на генерацію через API сумісне з “OpenAI” API. Якщо користувач хоче запускати ШІ на власному комп’ютері він може розгорнути у себе такий сервер і підключити до нього чат клієнт розроблений для роботи з віддаленими провайдерами.

Через наведені вище причини розроблювана програма написана саме з використанням віддалених провайдерів.

1.3 OpenAI API

Щоб програма могла працювати з віддаленим сервісом вона повинна імплементувати його API, що мож бути значною проблемою у розробці оскільки імплементация кожного API потребує зусиль.

На щастя, перший великий у цій галузі – компанія “OpenAI” надавала свої послуги через відкрите і задокументоване API яке називають “OpenAI API”. Це API призначене для використання власних сервісів компанії “OpenAI”, але більшість конкурентних провайдерів використовують це саме або подібне API для сумісності. Тому щоб охопити велику кількість можливих провайдерів клієнт-програми насправді достатньо імплементувати лише це API.

1.4 Огляд провайдерів що працюють через OpenAI API

У списку провайдерів що були розглянуті під час пошуку інформації були:

“OpenAI”, “xAI”, “DeepSeek”, “Novita AI”, “Antropic”, “Misrtal AI”, “LLM Gateway”.

Для тестування програми використовувався провайдер “OpenRouter”. На рис. 1.4 наведено зображення головної сторінки провайдера.

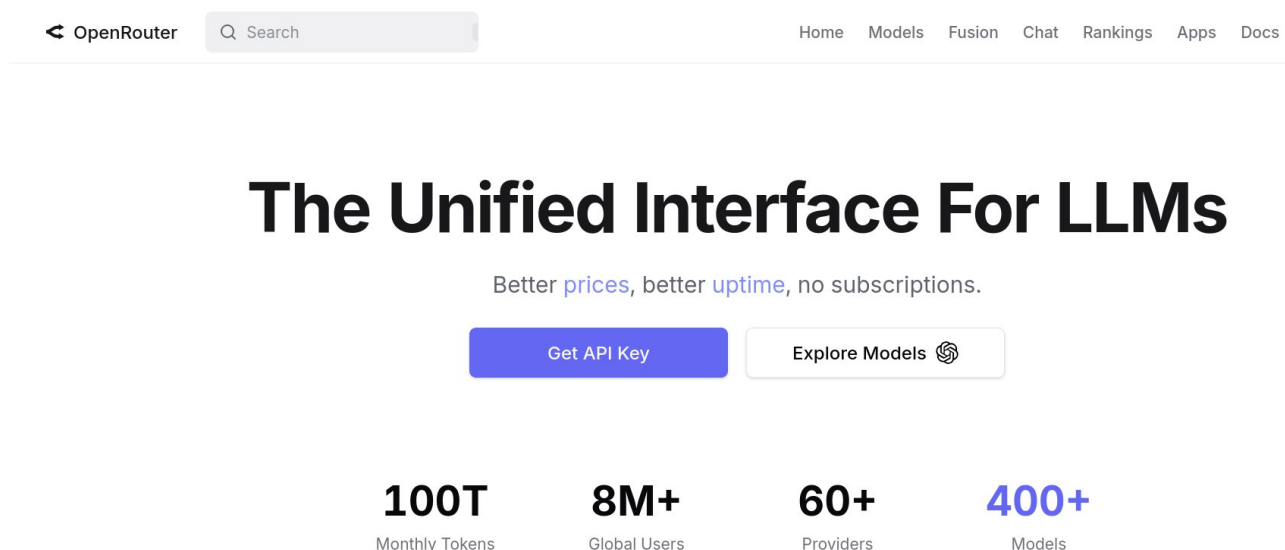


Рис. 1.4 Головна сторінка провайдера текстового ШІ “OpenRouter” за адресою “https://openrouter.ai”.

Цей провайдер не використовує машини висновування самостійно, а виступає посередником між клієнтом та провайдерами що це роблять. Це дозволяє використовувати багато моделей із одного API.

1.5 Огляд наявних чат клієнтів для текстового ШІ

“LM Studio” працює із машиною висновування “llama.cpp” яку програма встановлює та запускає самостійно. Програма надає можливість завантажувати моделі прямо з її інтерфейса (показано на рис. 1.5) або додавати моделі завантажені користувачем.

Після завантаження моделі програма використовує машину висновування для генерації тексту. Машина може використовувати графічну карту на користувацькому пристрої для прискорення генерації. Рис. 1.6 показує чат у програмі.

У цій програмі файли моделей або лежать у файлах програми, або керуються вручну користувачем, це займає багато місця на диску і вимагає більшої роботи від користувача.

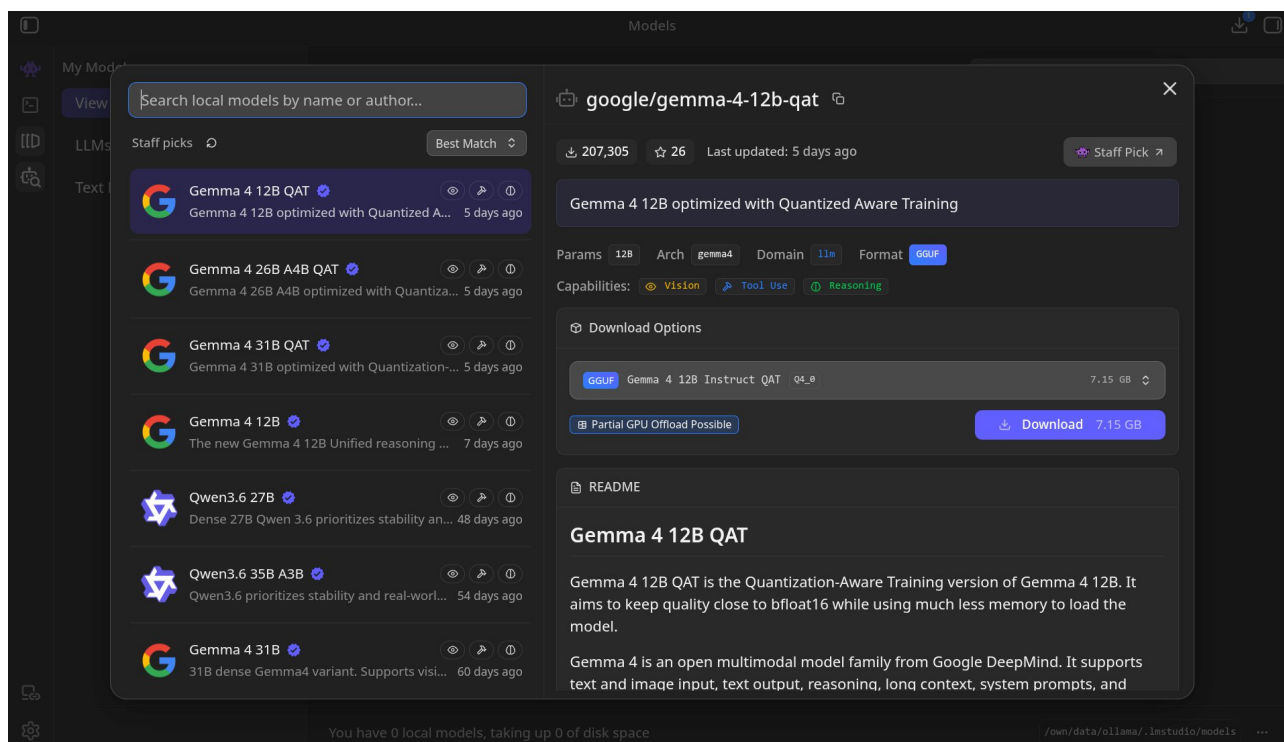


Рис 1.5 Список моделей доступних до завантаження у інтерфейсі програми “LM Studio”

Також у цій програмі є лише один вибір машини висновування, використовувати іншу або від віддаленого провайдера у цій програмі не можна.

“OpenWebUI” є веб додатком, тобто HTTP сервером з якого можна отримати графічний інтерфейс у браузері.

Ця програма не має власної машини висновування, а підключається до віддаленого провайдера через “OpenAI” API, зображено на рис. 1.8.

Програма розрахована на роботу на сервері із багатьма користувачами підключеними одночасно тому має систему користувачів та автентифікації, зображену на рис. 1.7.

Програма питає у провайдера про наявні моделі через API, як зображено на рис 1.9.

Після чого користувач може використати обрану модель для генерації тексту через API, як зображено на рис 1.10.

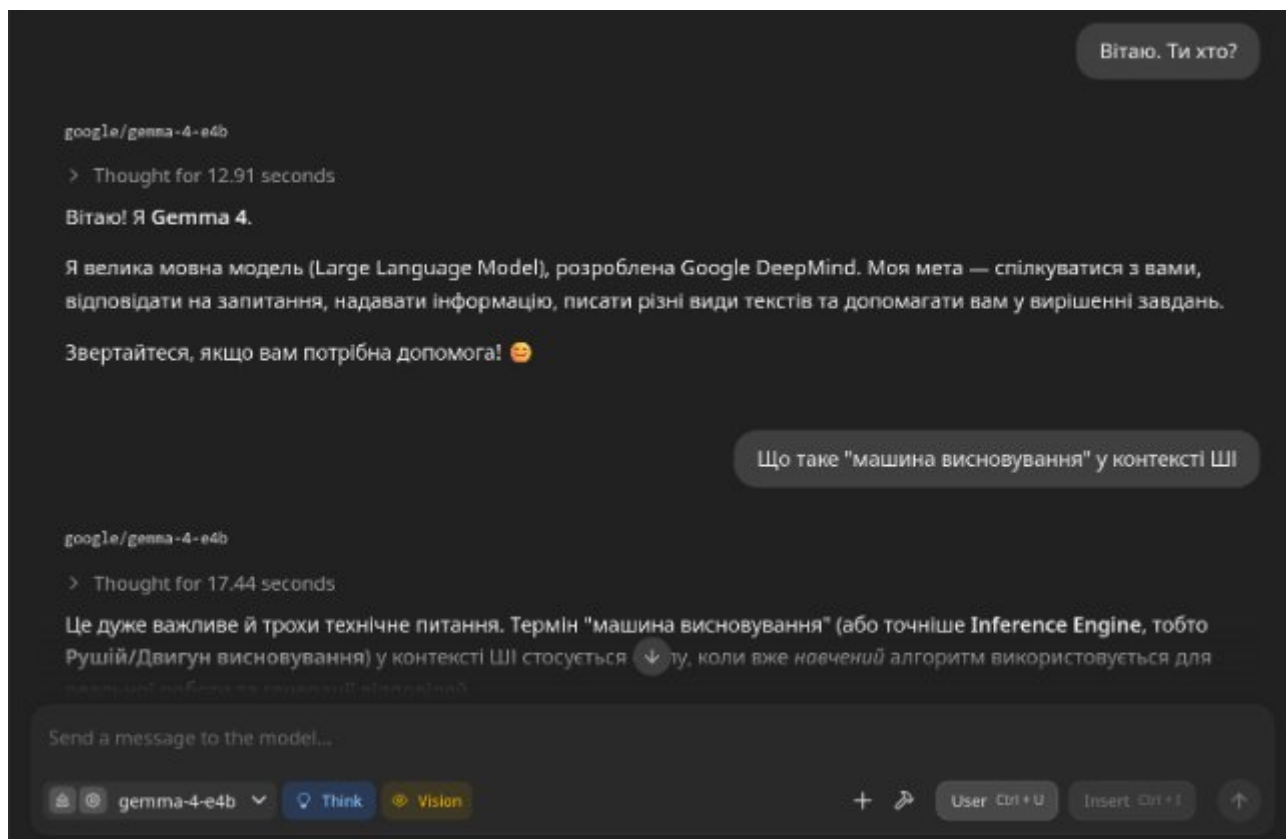


Рис 1.6 Чат із квантизованою моделлю “Gemma-4” у програмі “LM Studio”.

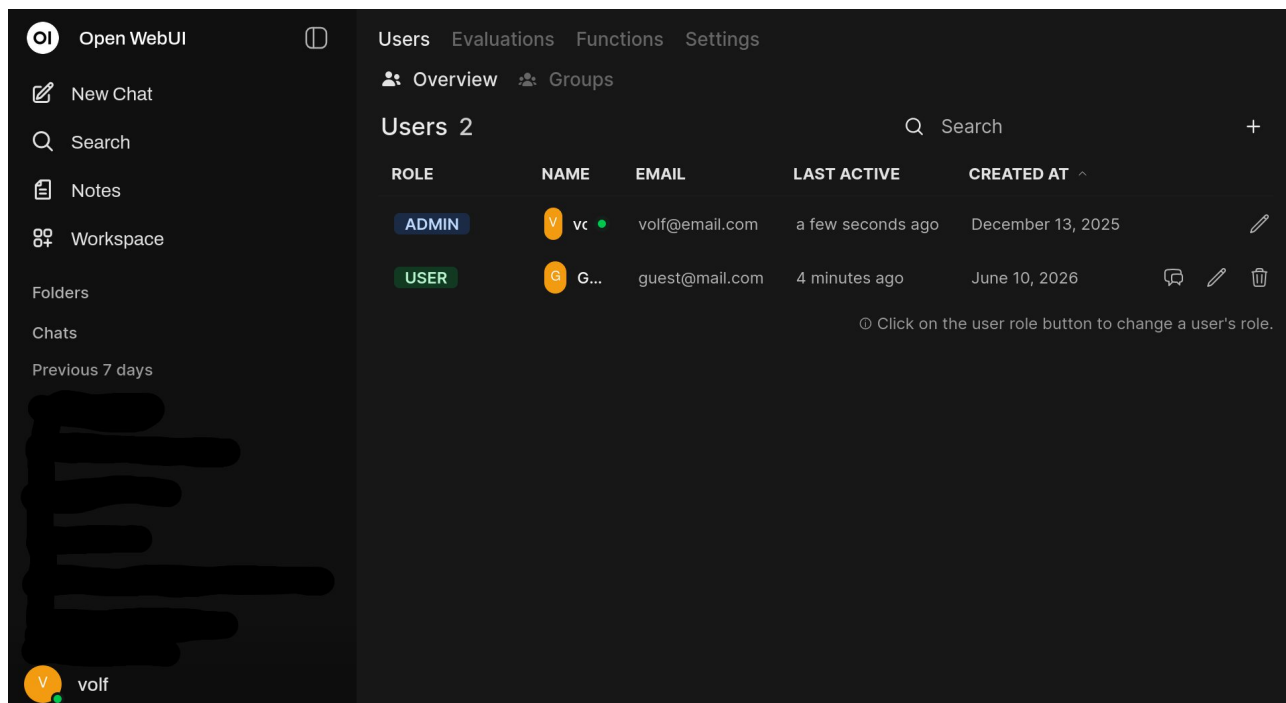


Рис 1.7 Панель керування користувачами у веб інтерфейсі у програмі “OpenWebUI”.

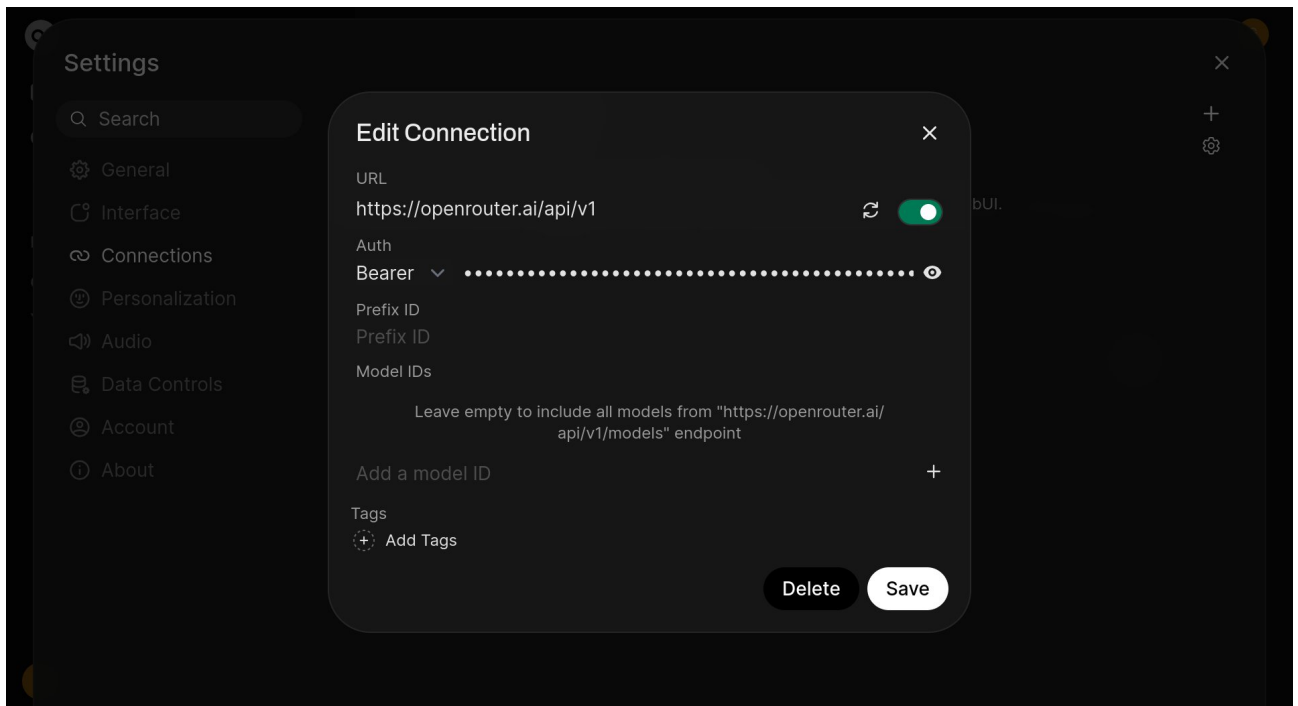


Рис 1.8 Користувач вводить дані потрібні для підключення до “OpenAI” API у веб інтерфейсі програми “OpenWebUI”.

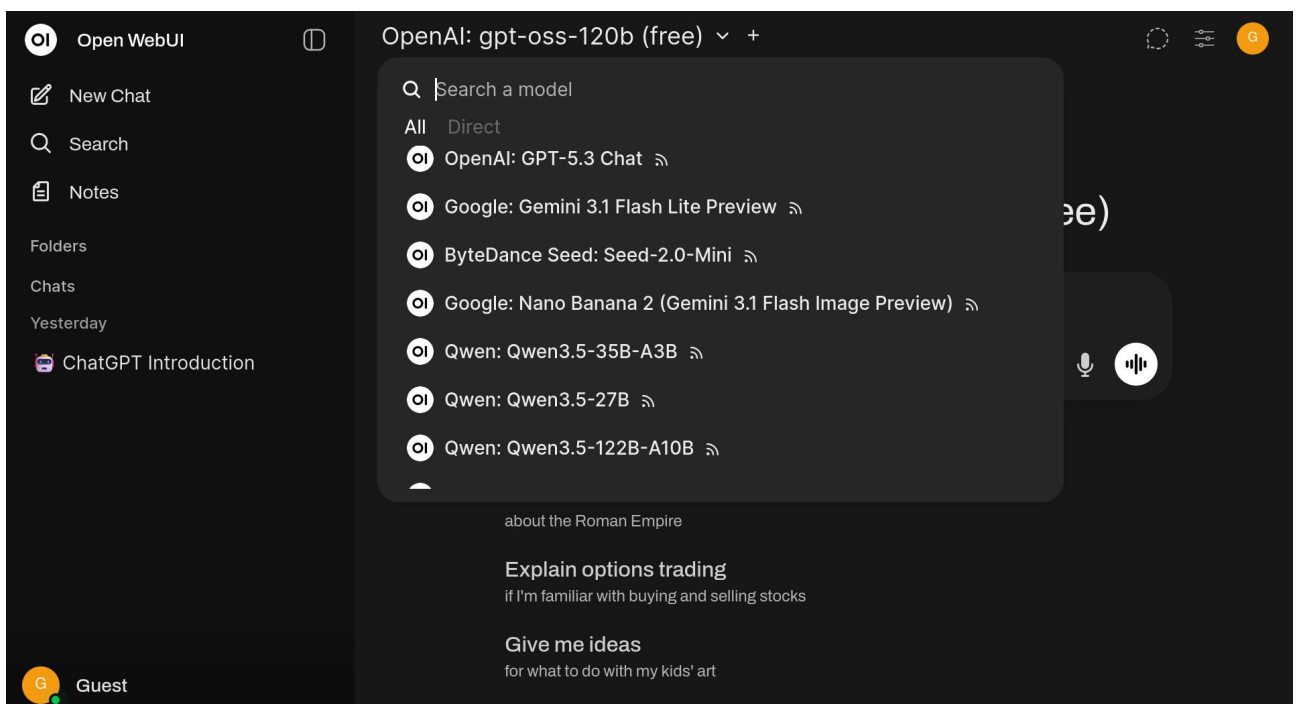


Рис 1.9 Веб інтерфейс програми “OpenWebUI” показує користувачу список моделей доступних у провайдера API.

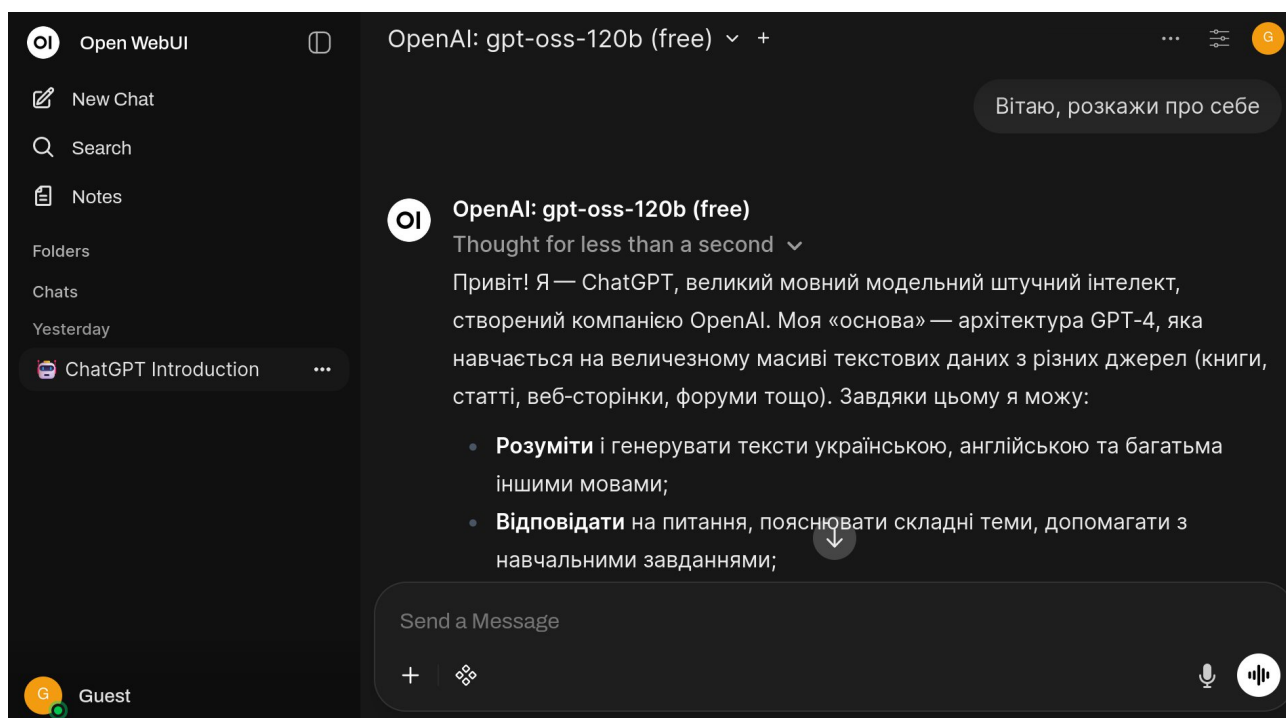


Рис 1.10 Відповідь згенерована моделлю “gpt-oss” у веб інтерфейсі програми “OpenWebUI”

Оскільки програма використовує віддалене API користувачу не треба тримати файли моделей у себе на пристрої.

Через те що ця програма є веб сервером користувачу може бути складно і страшно її встановлювати.

Розроблювана програма працює через “OpenAI” API, тому її інтерфейс користувача схожий на інтерфейс програми “OpenWebUI”.

Розроблювана програма є програмою віконною програмою, тому поширюється через файл для як програма “LM Studio”.

2 ПОШУК, ПОРІВНЯННЯ ТА ВИБІР ІНСТРУМЕНТІВ ТА МЕТОДІВ РОЗРОБКИ

2.1 Бібліотека для графічного інтерфейса користувача

Розроблювана програма є графічною віконною програмою. Робота графічної програми є достатньо складною. Спочатку програма має звернутися до віконного менеджера і отримати вікно, через його ж API програма просить віконного менеджера надсилати їй сигнали коли стаються переривання. Щоб намалювати інтерфейс програма має використати API драйвера графічної карти (або операційної системи). Під час отримання сигналів про переривання програма має обробити кожен з них щоб зрозуміти яку дію в інтерфейсі робить користувач, після чого використати знову ж графічне API щоб намалювати оновлений інтерфейс. Це все багато роботи. Тому було написано бібліотеки які абстрагують та спрощують цей процес.

Для малювання графіки існують графічні бібліотеки та системні API такі як “OpenGL”, “Vulkan”, кілька “DirectX”, “Metal” які абстрагують API драйверів різних виробників та операційних систем. Ці бібліотеки мають різну підтримку на різних платформах або інші проблеми тому існують абстракції над ними такі як “WebGPU”, “NVRHI”, “bgfx”.

Однак використання графічних бібліотек як і їх абстракцій потребує окремих графічних навичок оскільки їхнє використання по суті є програмуванням віртуальної графічної карти відносно низькорівневою мовою шейдерів, аналогією до цього процесу можна навести написання програм для центрального процесора за допомогою асемблера. Це багато роботи тому є бібліотеки що абстрагують вже над цими бібліотеками – ігрові двигуни чи бібліотеки рендерингу фігур.

Навіть якщо використовувати високорівневі бібліотеки для рендерингу фігур програмі треба складати складний інтерфейс із цих простих фігур що теж багато роботи.

Через це все існують ті бібліотеки які будуть розглянуті у цьому пункті – бібліотеки графічного інтерфейса користувача.

В процесі підбору було розглянуто чотири бібліотеки: “Tauri”, “Dioxus”, “Iced”, “Qt”, “GTK4” (та бібліотека мови Rust “gtk4” для сумісності), та “Relm4” яка врешті була обрана.

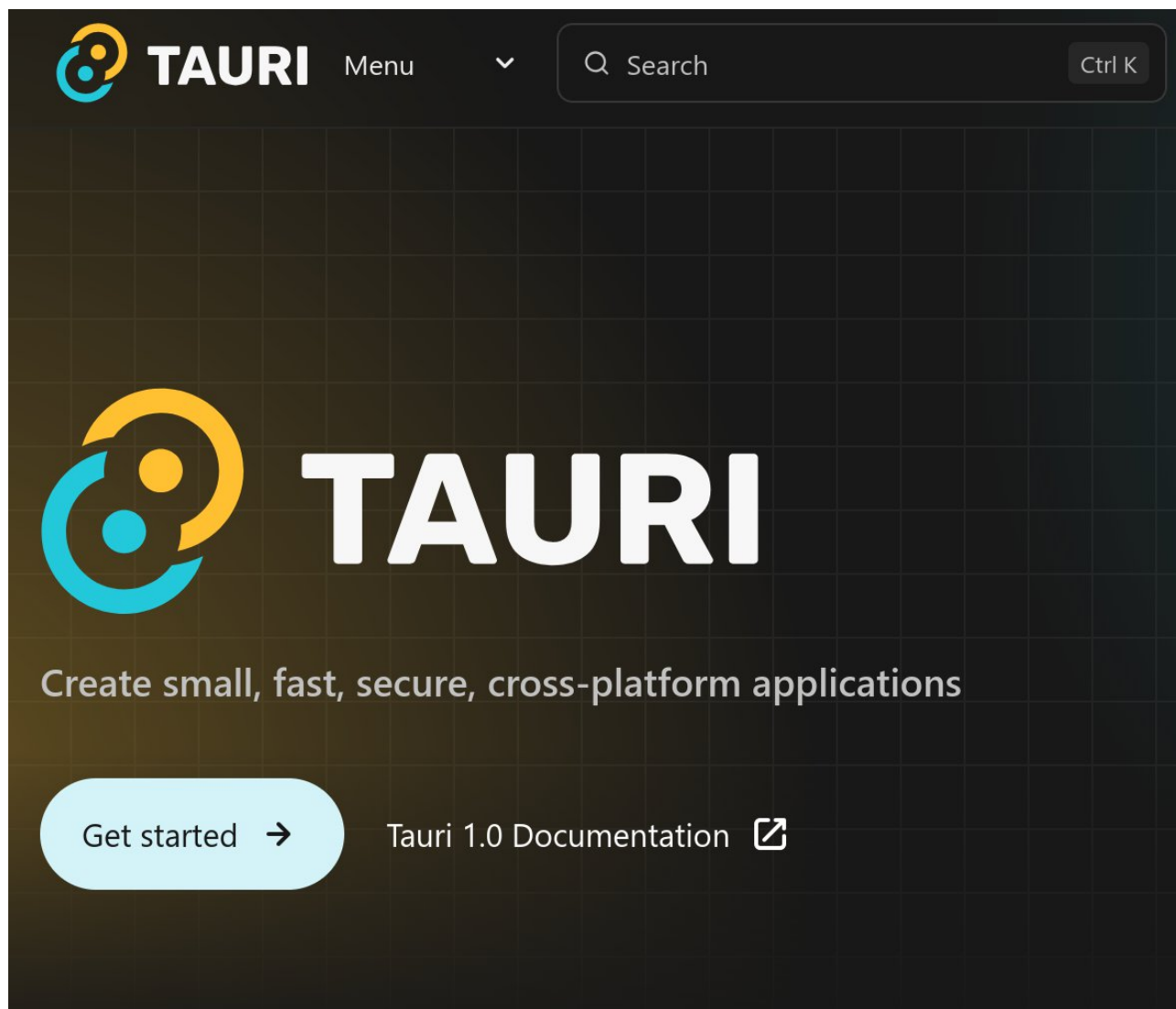


Рис 2.1. Головна сторінка веб сайта фреймворка “Tauri” за адресою “https://v2.tauri.app/”.

“Tauri” це вебпереглядач загорнутий у бібліотеку який надає можливість писати графічний інтерфейс як веб програму за допомогою веб технологій (HTML/CSS/JS) та будь яких веб фреймворків.

Розробник пише по суті дві програми – веб програму, таку як для браузера, та програму на мові Rust яку називають бекендом. Обидві частини знаходяться всередині одного виконуваного файлу яким програма поширюється.

Під час запуску програми вона запускає дочірній процес із системним веб переглядачем – віконною програмою яка інтерпретує веб програму.

Бібліотека налаштовує спеціальне API для веб програми яке дозволяє їй викликати функції з основної програми на Rust із передачею їм даних перед запуском і отриманням даних після їх завершення.

Таким чином програма працює у двох окремих процесах на двох окремих технологіях, де веб програма керує інтерфейсом а програма на Rust виконує системні операції.

Причиною існування цієї технології є те що використання власних графічних бібліотек для системних мов програмування сильно відрізняються від написання веб програм. Також веб програми це найпоширеніша галузь застосування графічних користувацьких інтерфейсів, цими технологіями володіє більша кількість людей, про них більше інформації та ці технології викладалися у цьому університеті.

Оскільки веб програми працюють у досить обмеженому середовищі браузера вони не можуть комунікувати із API операційної системи та виконувати системні операції, такі як зберігання та читання файлів. Тому для написання віконних програм за допомогою веб технологій потрібні допоміжні бібліотеки чи API. Фреймворк “Electron”, де весь код пишеться мовою “Java Script”, використовує інтерпретатор “Node.js” який має API для комунікації із операційною системою. Фреймворк “Tauri” працює як бібліотека для системної мови програмування тому вже має доступ до системних API.

У “Tauri” веб програму можна писати за допомогою будь яких браузерних веб технологій що видають статичні файли: ручною версткою, прогресивними “Java Script” фреймворками, технологіями що транслюють третій код у “HTML/CSS/Java Script” нахшталт транслювати “Java” у “Java Script” або “SCSS” у “CSS”.

Проблема фреймворка “Tauri” це різна система типів даних у мовах “Java Script” та “Rust”. “Java Script” використовує динамічну типізацію зі збирачем сміття

в той час як Rust має статичну типізацію і не має визначеного ABI. Щоб передавати дані між двома частинами програм “Tauri” використовує серіалізацію JSON об’єктів. На веб стороні дані що мають бути надіслані на сторону основної програми сереалізуються у текст у форматі “JSON”, бібліотека “Tauri” отримує цей текст і пробує десеріалізувати його у відповідний тип “Rust”. Коли основна частина готова повертати відповідь процес йде у зворотньому напрямку – вихідні дані із функції “Rust” серіалізуються у текст у форматі JSON який надсилається веб програмі. Коли веб програма отримує відповідь бібліотека пробує сереалізувати отриманий текст у об’єкт який програма може використовувати. У двох частинах програми одні й ті ж дані існують у двох різних формах тому операції над даними що можуть бути виконані в одній частині не можуть бути виконані в іншій. Також втрачається типізація з “Rust” яка робить розробку надійнішою. Це робить архітектуру програми складнішою і важчою до імплементації.

Також проблемою є продуктивність. Код мови “Java Script” інтерпретується. Також програму сповільнюватимуть постійні перетворення даних із та у “JSON” формат. Через ці причини фреймворк “Tauri” був відкинутий.

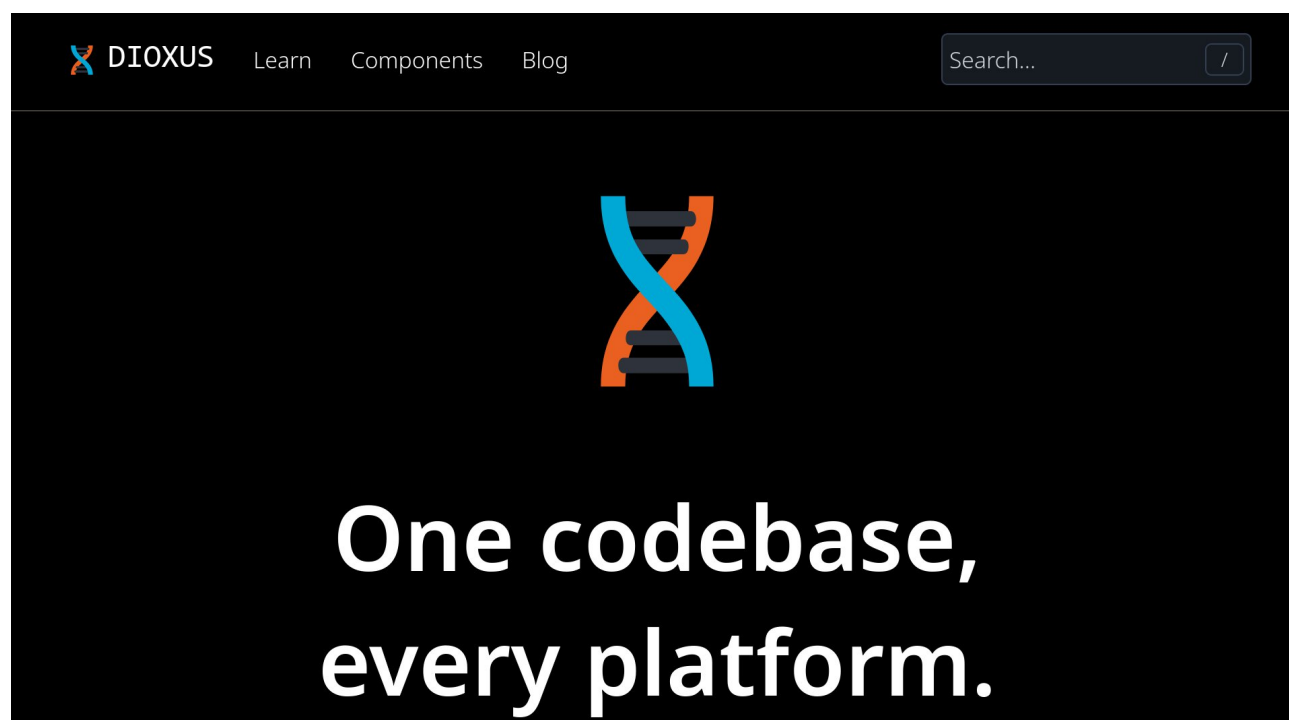


Рис 2.2 Головна сторінка фреймворка “Dioxus” за адресою
“<https://dioxuslabs.com/>”.

Розробник пише всю програму мовою “Rust”. Інтерфейс користувача будується об’єктами типа “Element” які в собі містять текст із розміткою “HTML”. Ці елементи конструюються спеціальними функціями визначеними користувачем і використовуються фреймворком програмно для окреслення інтерфейса. Фреймворк також надає макрос “rsx” здатний генерувати необхідний код для створення “HTML” елементів з тексту спеціального формату схожого на “HTML”, як це показано на рис 2.3.

```
#[component]
fn App() -> Element {
    rsx! {
        div { id: "title",
            h1 { "HotDog! 🌭" }
        }
        div { id: "dogview",
            img { src: "https://images.dog.ceo/breeds/pitbull/dog-3981540_1280.jpg" }
        }
        div { id: "buttons",
            button { id: "skip", "skip" }
            button { id: "save", "save!" }
        }
    }
}
```

Рис 2.3 Приклад функції для створення елемента з допомогою макроса “rsx”.

Із сторінки за адресою “<https://dioxuslabs.com/learn/0.7/tutorial/rsx>”.

На відміну від “Tauri” де веб програма збережена у вигляді статичних файлів які всі видаються інтерпретатору під час запуску “Dioxus” працює як “HTTP” сервер із рендерингом на сервері який надсилає “HTML” елементи до графічного інтерфейса по ходу виконання програми.

“Dioxus” надає приємніший програмний контроль за інтерфейсом користувача та усуває проблему несумісності типів. Оскільки “HTML” фрагменти генеруються на сервері, він може вшивати у них дані які він хоче показати користувачу під час створення елемента. У коді макроса “rsx” це виглядає як текстовий літерал що містить назву локальної “Rust” змінної оточену заокругленими дужками, як

“{variable}”; такий літерал буде замінений на тестову інтерпретацію змінної під час створення елемента.

Порівняно з “Tauri” “Dioxus” має один малий. У “Dioxus” не зручно писати веб програму браузерними фреймворками чи технологіями що транслюють третій код у “HTML/CSS/JavaScript”. Читачу що ознайомлений з фреймворком “Dioxus” може здатися що це взагалі не можливо, але насправді це просто не зручно; це можна зробити якщо писати статичний веб код із вставками літералів для заміни окремо щоб потім вручну імперативно перетворювати його у елементи.

Від ідеї використовувати веб фреймворки було відмовлено ще під час розгляду “Tauri”, а “Dioxus” із простим “HTML” не так сильно й відрізняється від бібліотек нативних до системних мов програмування, тому цей фреймворк було відкинуто в їх користь.

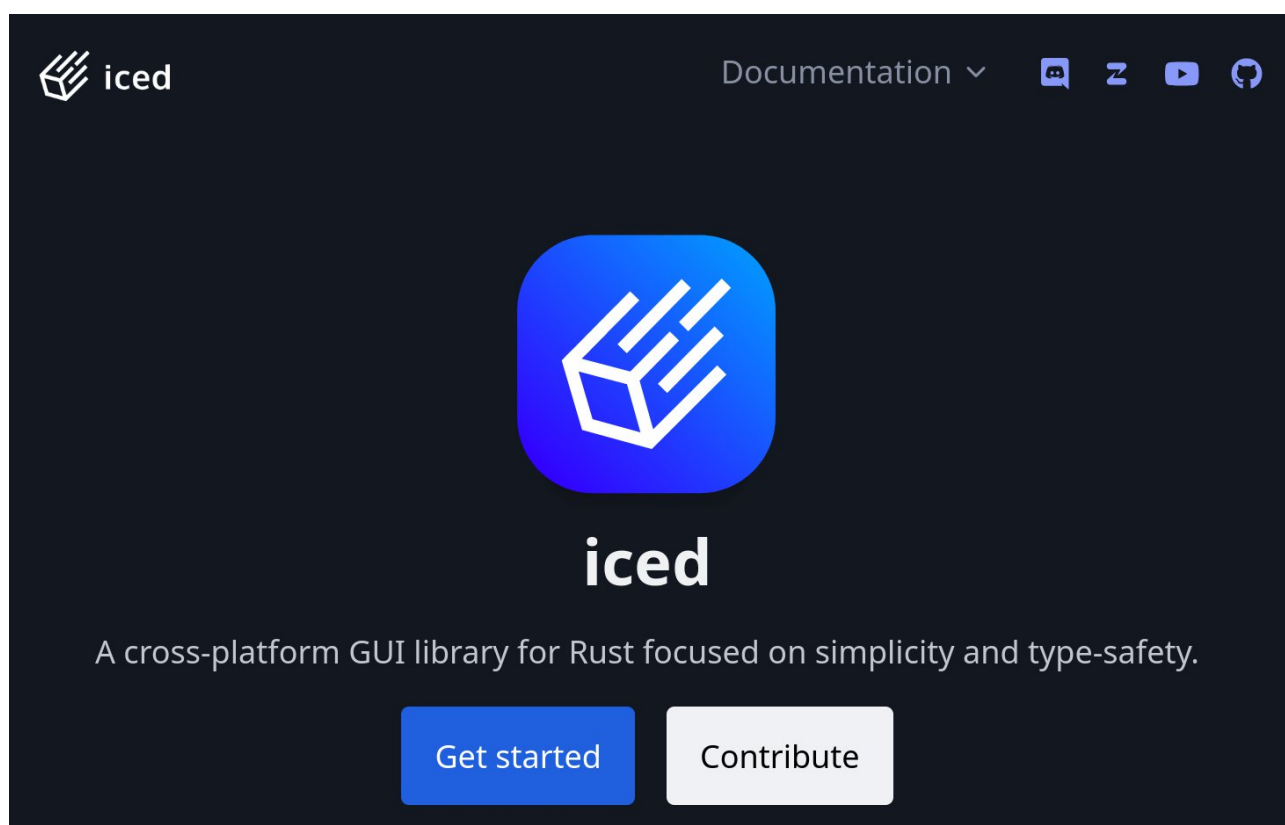


Рис 2.4 Головна сторінка бібліотеки “Iced” за адресою “<https://iced.rs/>”.

“Iced” це нативна “Rust” бібліотека. Вона створена щоб відтворити філософію архітектури графічного інтерфейса із мови програмування “Elm”. У цьому підході

елемент графічного інтерфейса складається із трьох частин – віджетів які цей елемент показує, стану - даних які використовують ці віджети, та обробки подій які ці віджети можуть відправити на обробку.

Для того щоб намалювати віджети програма дає їм початковий стан. Для цього елемента визначено функцію яка обробляє подію, ця функція приймає на вхід повідомлення. Віджети програмуються відправляти певне повідомлення під час їхніх подій; це повідомлення від віджета отримує функція обробки події яка в залежності від повідомлення вносить зміни до стану. Внесення змін до стану автоматично спричиняє оновлення віджетів і перемалювання графіки.

Ця модель дуже цікава і приємна, однак від використання цієї бібліотеки було відкинуто через сумніви щодо її якості. Проект відносно новий і пишеться однією людиною, через що може мати багато непередбачуваних проблем і багів, про що відверто написано у документації цієї бібліотеки.

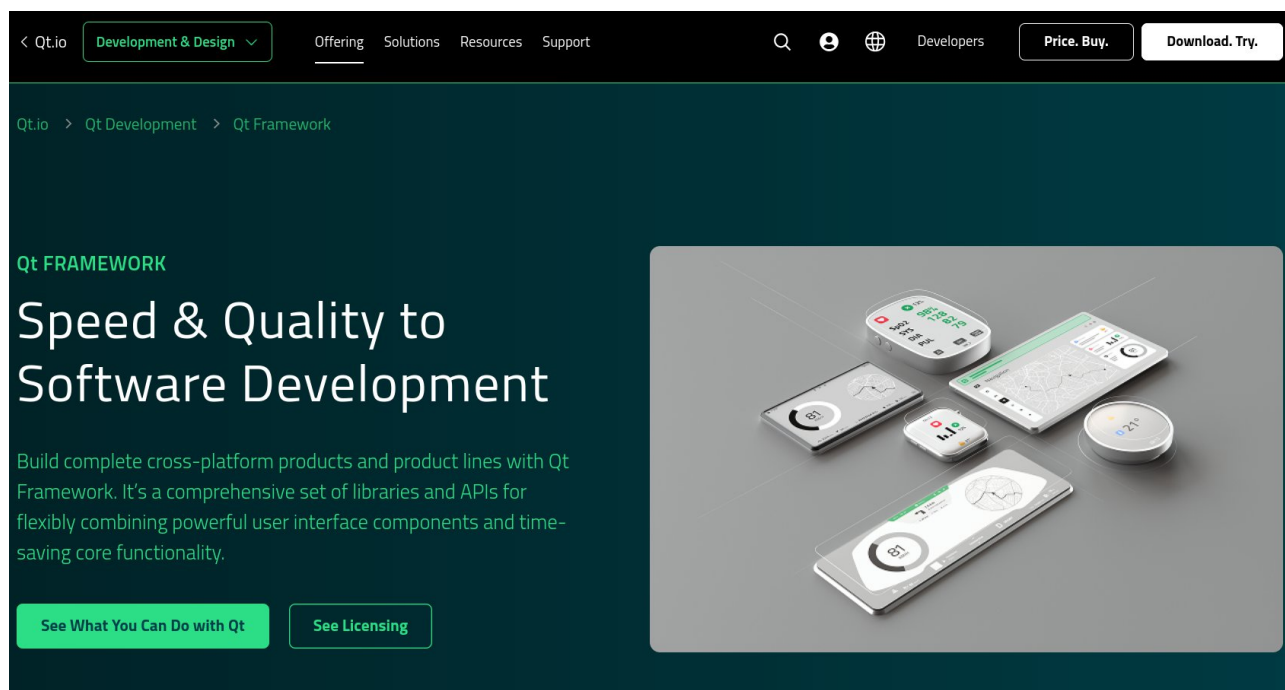


Рис 2.5 Головна сторінка фреймворка “Qt” за адресою
“<https://www.qt.io/development/qt-framework>”.

“Qt” це великий комерційний фреймворк для C++ який використовується у тонні програмного забезпечення. Фреймворк імперативний, де контроль над вікном та віджетами виконується за допомогою функцій.

Попри те що фреймворк комерційний його ліцензія дозволяє його використання у програмах з відкритим кодом і деякими іншими умовами.

Цей фреймворк був мало досліджений і відкинутий рано через те що він має більший поріг входу ніж інші графічні бібліотеки, а на час його розгляду розробка вже велась поспіхом. До того він схожий на бібліотеку “GTK”, якій було надано перевагу.

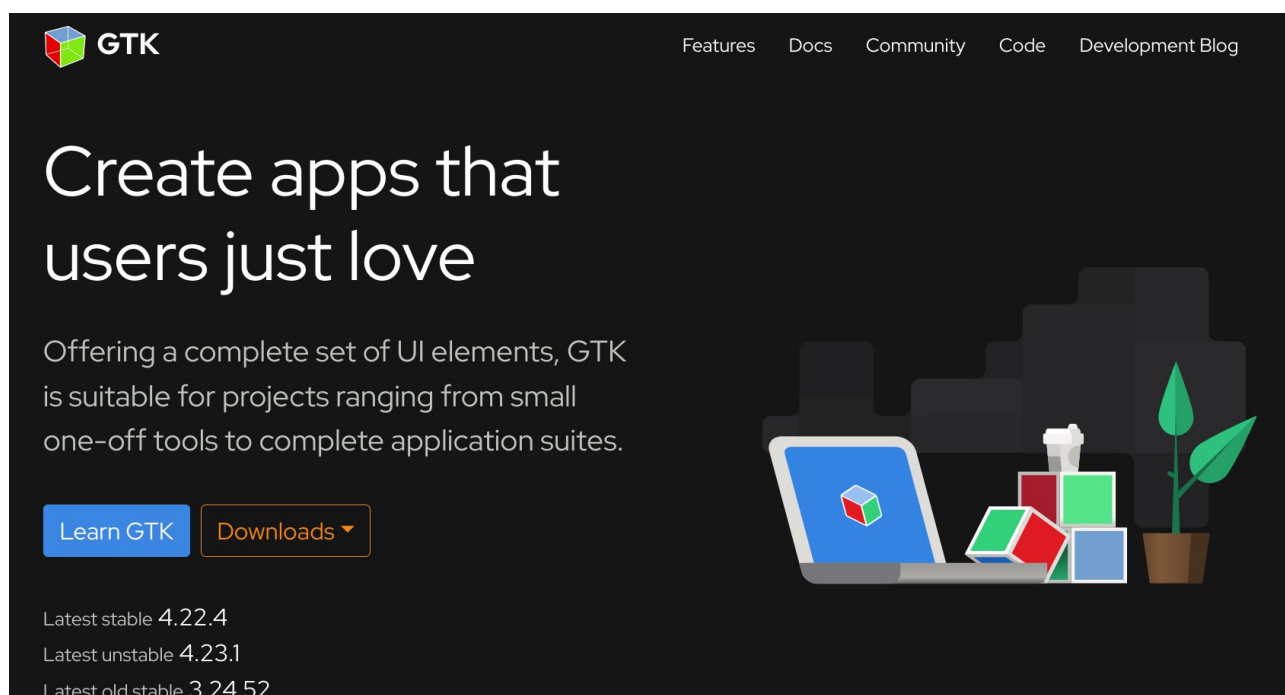


Рис 2.6 Головна сторінка проекту “GTK” за адресою “<https://www.gtk.org/>”.

Бібліотека “GTK4” із серії бібліотек “GTK”, що означає “GIMP toolkit”, була розроблена для розробки растрового графічного редактора “GIMP” проекту “GNU”.

Бібліотека є імперативною бібліотекою для мови програмування “C”. Бібліотека надає можливість запросити створення вікна та його керування. Після створення вікна програма може імперативно малювати у ньому віджети за допомогою функцій. Кожен віджет має набір подій які можуть із ним статися, які відповідають діям які користувач може виконати у графічному інтерфейсі. Для

кожного віджета визначається окрема функція для обробки окремої події яка запускається коли подія стається.

Ця бібліотека існує давно, розробляється відомою організацією та є надійною.

Ця бібліотека використовується у розроблюваній програмі але не напряму.

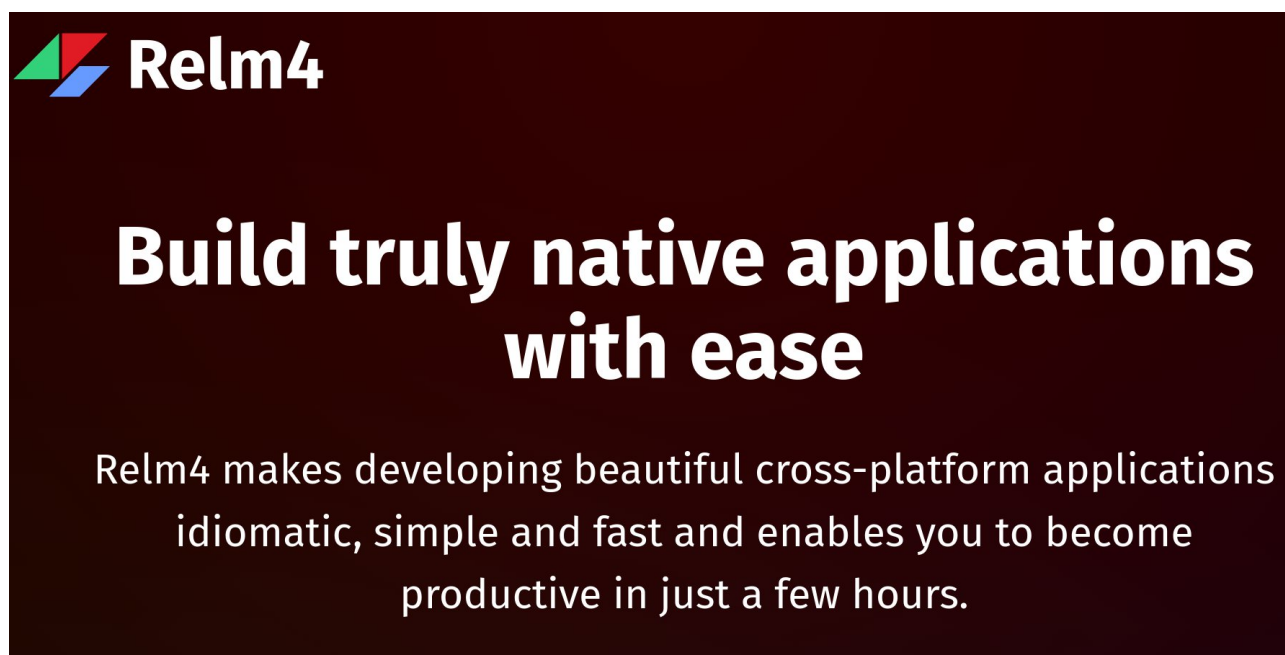


Рис 2.7 Головна сторінка бібліотеки “Relm4”, за адресою “<https://relm4.org>”.

Бібліотека “Relm4” це невелика надбудова над бібліотекою “GTK4” яка використовує віджети “GTK” щоб будувати інтерфейс за методикою мови “Elm”. Ця бібліотека працює із окремими модулями користувацького інтерфейса кожен з яких називається “компонент”. Компонентом є будь яка структура створена розробником що імплементує мовний інтерфейс “Component”, ця структура містить дані що складають “стан” у моделі мови “Elm”. Віджети інтерфейса користувача визначаються у функціях компонента “init”, та “init_root”. Для кожного віджета “GTK” можна визначити функцію яка виконуються коли із цим віджетом стається подія. Замість того щоб виконувати логіку змін у цих функціях вони натомість надсилають повідомлення до однієї функції “update” визначеної для компонента, яка вже імплементує логіку в залежності від повідомлення.

Ця бібліотека імплементує не все так як у методиці “Elm”. Коли в стан компонента вносяться зміни віджети не оновлюються самостійно. Навіть попри це така архітектура допомагає розбивати інтерфейс на модулі та полегшити розробку.

2.2 Вибір способу зберігання даних

Програма працює із даними які треба зберігати коли програма завершується – текст чатів та налаштування програми, тому програма потребує способу зберігати їх на диск

Вибір проводився між двома методами зберігання даних: у серіалізованому вигляді у файлах що керуються вручну або у локальній базі даних.

Просто кажучи база даних це організована структура даних для якої є програма або бібліотека якою із цією структурою даних можна працювати: отримувати дані із неї, записувати у неї дані, шукати дані за спеціальними характеристиками. Чи є база даних реляційною чи ні не є важливо для цього проекту тому що йому потрібні лише прості операції збереження та отримання даних які всі бази даних і так імплементують, тому для проекту розглядалися всілякі бази даних.

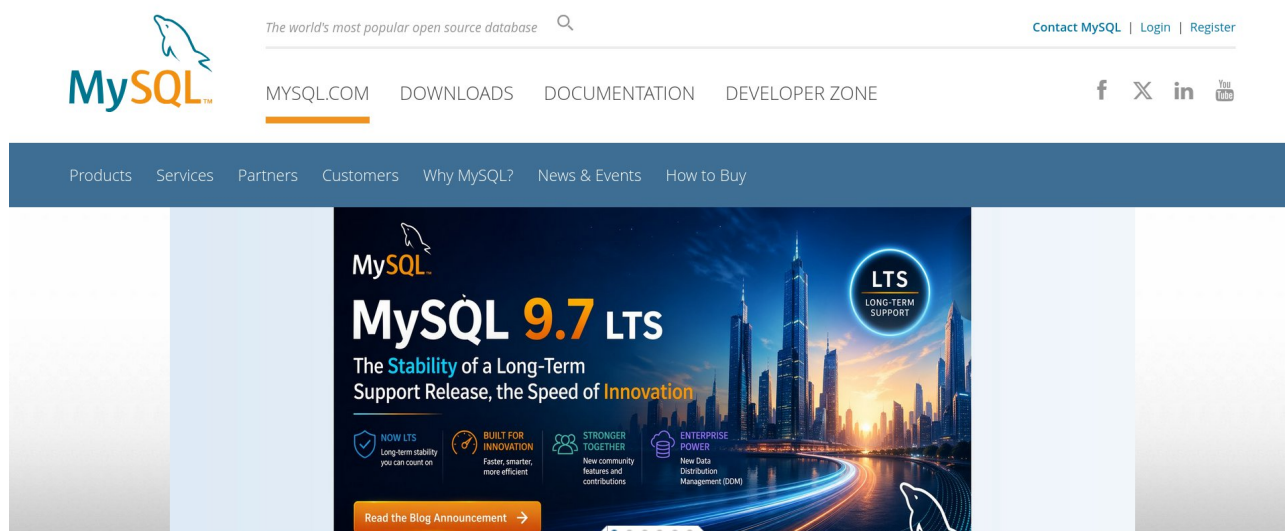
Було розглянуто наступні бази даних: “MySQL Community Edition”, “MariaDB”, “SQLite”, “MongoDB Community Edition”.

“MySQL” це комерційна реляційна SQL база даних компанії “Oracle” що має безкоштовну версію “Community Edition”.

Поширюється і працює “MySQL” у вигляді TCP сервера який тримає базу даних у файлах на диску комп’ютера на якому він працює у вигляді реляційних таблиць, і тримає API через яке автентифіковані та авторизовані користувачі можуть виконувати операції над даними надсилаючи відповідні команди мовою SQL.

Хоч ця база даних розроблена як сервер розроблювана програма може використовувати її якщо розверне сервер локально.

Ця база даних надає великий рівень контролю над запитами і робить складну фільтрацію даних ще на стороні сервера перед їх надсиланням програмі, але оскільки програма цього всього не потребує це не є перевагою.



MySQL HeatWave

Use automated and integrated generative AI and machine learning (ML) in one cloud service for transactions and lakehouse scale



MySQL AI

MySQL AI provides integrated, automated, and secure machine learning (ML) and generative AI capabilities.

Рис 2.8 Головна сторінка бази даних “MySQL”, “<https://www.mysql.com/>”.

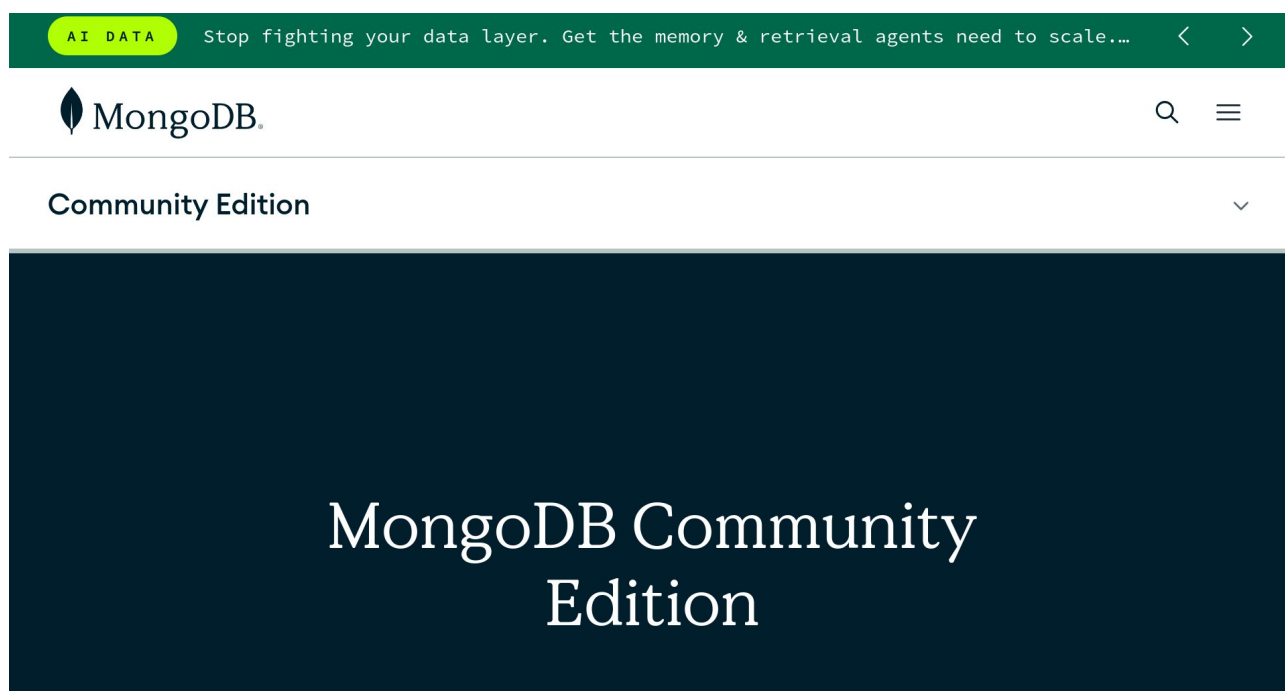


Рис 2.9 Головна сторінка бази даних MongoDB,
“<https://www.mongodb.com/products/self-managed/community-edition>”.

“MongoDB” це не реляційна однак типізована база даних яка так само як попередні дві розгортається як TCP сервер із автентифікацією та авторизацією. Ця

база даних відрізняється внутрішнім способом збереження даних, структурою даних та способом взаємодії із базою даних. У реляційних базах даних одиницею даних над якими виконується робота є рядок таблиці, яка має статичний тип відповідно всі рядки мають однакову структуру. “MongoDB” відрізняється тим що файли (які тут є аналогом рядка з реляційної бази даних) не мають типа, відповідно кожен з них у колекції може мати іншу структуру, однак це не заважає тримати в одній колекції виключно файли з однаковою структурою та мати зв’язки між ними як в реляційній базі даних. Для взаємодії із базою даних використовуються не інструкції SQL а набір функцій які містяться у бібліотеці для цієї бази даних які можна використовувати у програмі чи у “MongoDB Shell” – оболонкою що інтерпретує команди написані на “JavaScript”.



Рис 2.10 Головна сторінка бази даних “MariaDB”,
[“https://mariadb.com/products/community-server/”](https://mariadb.com/products/community-server/).

“MariaDB” це відгалуження від “MySQL” з відкритим кодом створене її колишніми розробниками через те що “MySQL” була придбана компанією “Oracle”. Через це ця база даних дуже схожа на “MySQL” тому її детального опису не буде –

все що стосується “MySQL” також стосується “MariaDB”. Єдиною різницею є їх розробники, власність та моральна боротьба спільноти відкритого коду.

“MariaDB”, “MySQL” та “MongoDB” всі є серверами, відповідно мають бути розгорнуті через локальний сервер для цих баз даних на комп’ютері на якому працює програма через що програмі потрібно мати власне файли серверів та правильно її запустити, тут починаються проблеми. Сама розроблювана програма пишеться мовою Rust, що робить її достатньо портативною, однак різні середовища на яких сама програма працюватиме однаково можуть вимагати різних файлів для сервера бази даних. Процес запуску сервера може відрізнятися на різних системах, а отже вимагатиме різного коду для кожного середовища. Це збільшить кількість роботи, складність розробки та збільшить ймовірність багів, тому ці три пропозиції були відкинуті.

“SQLite” це бібліотека для мови “C” що надає можливість створити та використовувати реляційну “SQL” базу даних у файлі. Ця база даних відрізняється тим що стовбці таблиць мають динамічну типізацію де кожне значення у ньому може мати будь який тип. Бібліотека має типи даних, але вони твердо дотримуються лише для конкретних значень. Динамічна типізація для бази даних є поганою ідеєю адже вона сповільнює роботу програми та збільшує ймовірність того що в будь який момент часу база даних містить не коректні дані, хоча це не є критичною проблемою.



*Small. Fast. Reliable.
Choose any three.*

[Home](#)
[About](#)
[Documentation](#)
[Download](#)
[License](#)
[Support](#)
[Purchase](#)
[Search](#)

SQLite is a C-language library that implements a [small](#), [fast](#), [self-contained](#), [high-reliability](#), [full-featured](#), SQL database engine. SQLite is the [most used](#) database engine in the world. SQLite is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day. [More Information...](#)

SQLite [source code](#) is in the [public-domain](#) and is free to everyone to use for any purpose.

Latest Release

[Version 3.53.2](#) (2026-06-03). [Download](#) [Prior Releases](#)

Common Links

- [Features](#)
- [When to use SQLite](#)
- [Getting Started](#)
- [Try it live!](#)
- [SQL Syntax](#)
 - [Pragmas](#)
 - [SQL functions](#)
 - [Date & time functions](#)
 - [Aggregate functions](#)
 - [Window functions](#)

Рис 2.11 Головна сторінка бази даних “SQLite”.

Таблиця 2.1

Порівняльна характеристика розглянутих баз даних

База даних	Потреба розгортати локальний сервер	Має визначені типи колекцій
“MySQL”	Так	Так
“MariaDB”	Так	Так
“MongoDB”	Так	Ні
“SQLite”	Ні	Ні

Після перегляду всіх варіантів баз даних було розглянуто спосіб зберігання даних у серіалізованому вигляді у файлах програми.

Порівняно із базою даних цей спосіб простіший в теорії – програма створює каталог у файловій системі у якому створює файл для кожного типу даних який треба зберегти, зазвичай структур. Серіалізація це процес перетворення даних із однієї форми запису в інший. Програма перетворює дані у формат для запису який вона зберігає у файлі. Потім коли програмі потрібні ці дані вона читає їх з файла та перетворює назад у структури в пам’яті.

Цей спосіб зберігання даних має бути імплементований програмою вручну, бо якщо для цього використовувати бібліотеку то він вже й не сильно відрізняється від варіанта бази даних (та і до якості такої бібліотеки будуть сумніви), однак цей спосіб імплементачії був обраний. Причиною є те що розробник вже працював із всіма раніше згаданими базами даних, а зберігання в звичайних файлах жодного разу не робив.

Після вибору файлів як способу зберігання питання було лише в тому яким чином та у який формат їх серіалізувати. Формати у які можна серіалізувати дані поділяються на бінарні та текстові. Текстові формати довше обробляються програмою та алгоритми їхнього парсингу складніші, але вони можуть бути легко прочитані та редаговані користувачем. Бінарні формати легше та швидше обробляються програмою та займають менше місця на диску але не можуть бути прочитані чи редаговані користувачем. Насправді бінарні файли могли би бути

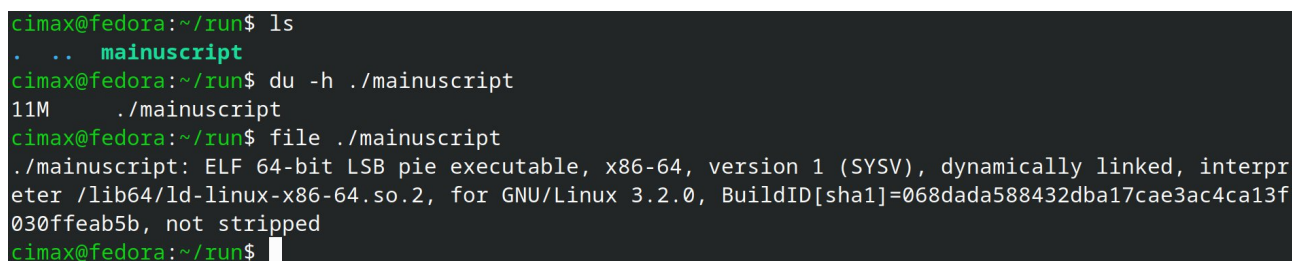
прочитані та редаговані користувачем якби користувач мав спеціальну програму для цього, так само як користувачу потрібен текстовий редактор для роботи з текстовими файлами, але програми для роботи з бінарними файлами мають бути імплементовані окремо для кожного типу файла щоб могли його розуміти щоб показати користувачу його вміст у вигляді у якому користувач може його розуміти, а текстові файли всі написані кількома відомими текстовими кодуваннями які легко інтерпретувати у зручному для користувача вигляді і редагувати. Наприклад зображення форматів “PNG” та “JPEG” є бінарними файлами, але вони є поширеними тому для них існують програми для їх перегляду та редагування...

Було прийнято рішення використати бінарну серіалізацію бо текстова серіалізація вже використовувалася розробником у попередніх проектах.

3 ОГЛЯД, ПРОЕКТУВАННЯ ТА ІМПЛЕМЕНТАЦІЯ ПРОГРАМИ

3.1 Огляд роботи програми

Програма “mAlnuscript” поширюється у вигляді бінарного виконуваного файлу. У цьому розділі розглянуто запуск та роботу програми на операційній системі “Fedora”. Програма розмішена на диску комп’ютера у файлі формату “ELF4” та готова до запуску, як показано на рис 3.1



```
cimax@fedora:~/run$ ls
. .. mainuscript
cimax@fedora:~/run$ du -h ./mainuscript
11M    ./mainuscript
cimax@fedora:~/run$ file ./mainuscript
./mainuscript: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=068dada588432dba17cae3ac4ca13f030ffeab5b, not stripped
cimax@fedora:~/run$
```

Рис 3.1 Вікно емулятора терміналу у якому видно виконуваний файл програми “mainuscript”, його розмір та формат

Після запуску програми швидко відкривається її вікно, яке показане на рис 3.2. У вікні програма малює графічний інтерфейс користувача із який користувач може взаємодіяти використовуючи комп’ютерну мишу та клавіатуру. На зображенні видно що інтерфейс складається з трьох частин – смужки зліва, що містить логотип і кнопки зверху і довгий відділ знизу зліва, та великої сірої зони.

Лівій відділ у межах проекту називається “навігація” а відділ справа “робоча область”.

Робоча область це місце де програма малюватиме корисний інтерфейс.

Зверху у навігації знаходяться 4 кнопки. Зелена кнопка “Settings” відкриває у робочій області секцію “Налаштування”, яка містить 2 поля для введення даних

необхідних для роботи “OpenAI” API. Коричнева кнопка “ListModels” надає можливість переглядати та обирати модель ШІ яка буде викопристовуватись для генерації тексту. Рядок із синьою кнопкою “New” та червоною кнопкою “Delete” надає можливість створювати та видаляти чати.

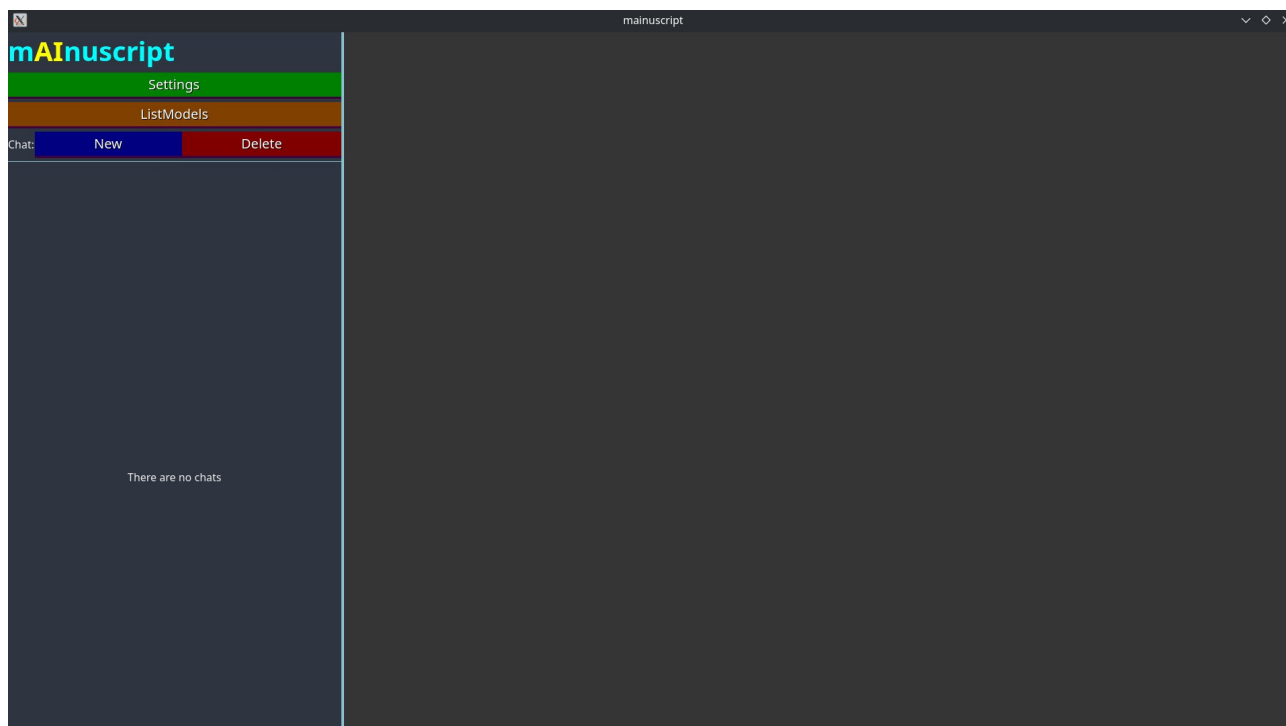


Рис 3.2 Вікно програми “mainuscript”

Зараз, одразу після першого запуску програма не має жодних чатів. Велика порожня область у навігації із надписом “There are no chats” це список у якому будуть кнопки що відкривають кожен чат.

Спочатку відкрито налаштування та введено туди адресу провайдера “OpenAI” API та токен для автентифікації.

Після натискання кнопки “Settings” у робочій області малюється інтерфейс налаштувань, як показано на рис 3.3

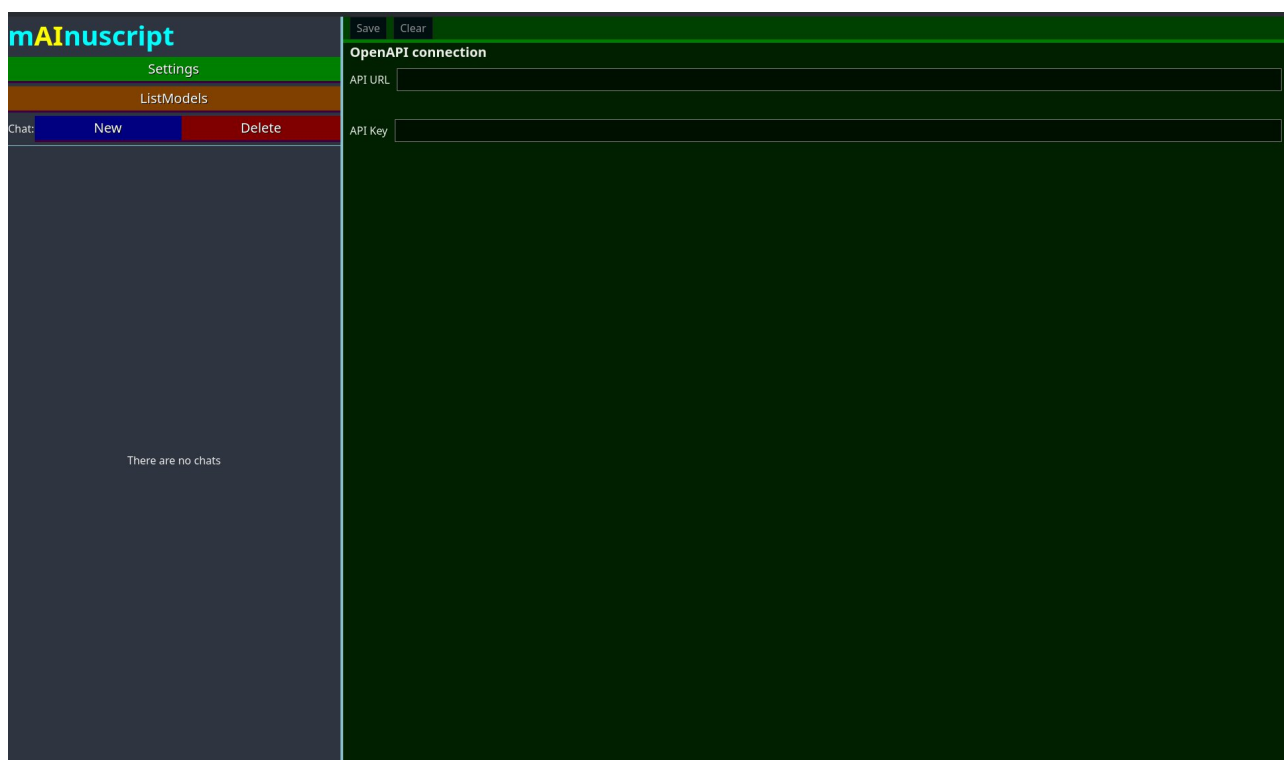


Рис 3.3 Щойно відкритий інтерфейс налаштувань у програмі “mainuscript”.

Інтерфейс налаштувань містить два текстових поля куди можна ввести текст після того як це поле натиснуто мишею. Спочатку у поля введено дані які мають неправильний формат – у перше поле введено “wrong_format”, а у друге поле введено “i refuse!”, як зображено на рис 3.4.

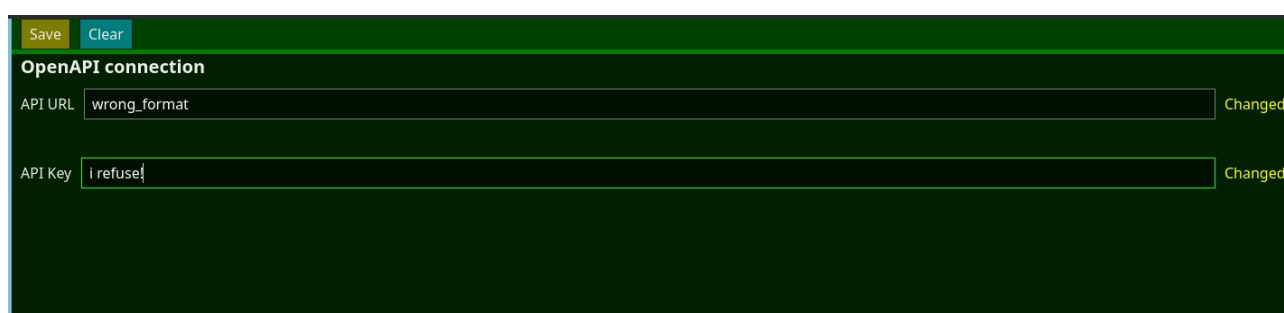


Рис 3.4 Тестові невірні значення введені у поля, справа від полів засвітілись мітки що вказують що введено нові значення.

Після введення першої букви у поле кнопки “Clear” та “Save” у горі засвітілися – їх тепер можна натискати. Кнопка “Clear” поверне поля введення до того стану у якому вони були до внесення змін. Кнопка “Save” спробує прочитати значення та

встановити їх у налаштування якщо вони коректні. Також після введення справа від кожного поля засвітилися мітки що вказують про те що у поле внесено зміни, ці мітки позначають поля що будуть повернені до попереднього стану після натискання кнопки “Clear”.

Далі було натиснуто кнопку “Save”, після чого угорі справа від кнопок з’явився текст із помилкою, зображено на рис 3.5. Помилка говорить про неправильний формат тексту введенного у поле “API URL”.



Рис 3.5 Програма вивела повідомлення про помилку у полі “API URL”.

Далі у поле “API URL” введено URL провайдера мовних моделей “OpenRouter”, а саме “https://openrouter.ai/api/v1”. Після цього знову натиснуто кнопку “Save”. Тепер повідомлення із помилкою змінилося на “Openai api key seems to be invalid”, як показано на рис 3.6.



Рис 3.6 Повідомлення про не валідний ключ API у програмі “mainuscript”.

Далі програму закрито. Після закриття програма зберегла текст у полях у файл “~/.mainuscript/”. Після того як програма була відкрита назад текст у полях був відновлений, як показано на рис 3.7.

Рис 3.7 Налаштування програми “mainuscript” після її перезапуску.

Далі введено коректний ключ. Після цього знову натиснута кнопка “Save”. Повідомлення “Changed” справа від полів зникли і кнопки знову не активні як показано на рис 3.8 – це означає що налаштування були встановлені.

Рис 3.8 Заповнені налаштування програми “mainuscript”.

Далі звернуто увагу до кнопок “New” та “Delete” у навігації. Натиснуто синю кнопку “New”, після чого інтерфейс програми змінився (рис 3.9).

Рис 3.9 Стан програми “mainuscript” після натискання кнопки “New”.

Текст кнопки “New” змінився на “Save”, і текст кнопки “Delete” змінився на “Cancel”, також під кнопками з’явилося текстове поле для введення. Програма виконує процедуру додавання нового чата. Кнопка “Cancel” скасує процедуру і поверне інтерфейс до попереднього стану без додавання нового чата. Далі у текстове поле введено “Перший” і натиснуто кнопку “Save”, одразу після чого кнопки повернулися до попереднього стану, текстове поле зникло а під ними з’явилася світло синя кнопка із надписом “Перший”.

Далі цю процедуру повторено і створено два інших чати: “Другий” та “Третій”, рис 3.11.

Далі натиснуто кнопку “Delete”. Кнопки одразу змінилися: синя кнопка “New” згасла щоб показати що вона не робочі а червона кнопка “Delete” стала сірою та має надпис “Cancel”, рис 3.12. Зараз програма знаходиться у процедурі видалення чата. Натискання кнопки “Cancel” скасує процедуру і поверне інтерфейс до попереднього стану. Натискання будь якого чата видалить його. Далі натиснуто чат “Другий” після чого він одразу зник а інтерфейс повернувся до попереднього стану як показано на рис 3.13.

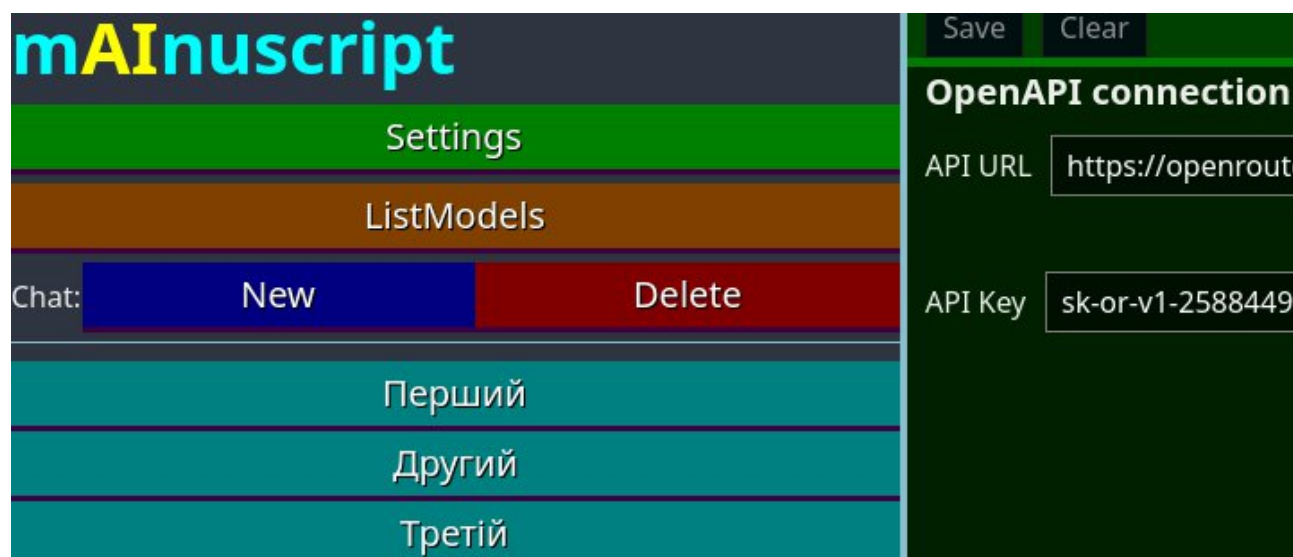


Рис 3.11 Три чати у програмі “mainuscript”

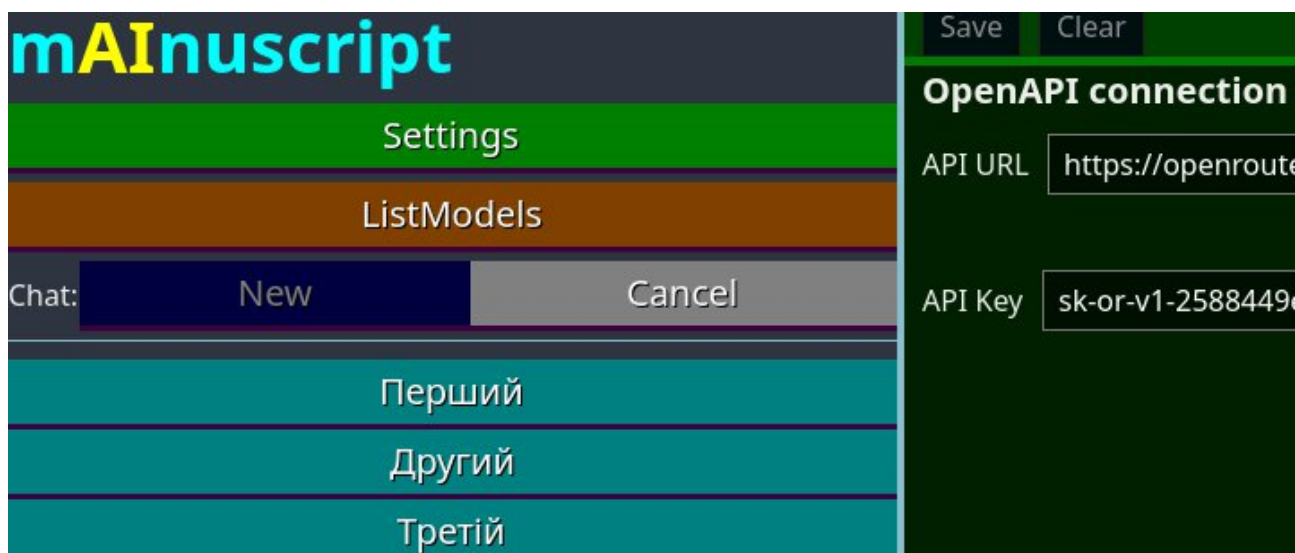


Рис 3.12 Програма “mainuscript” під час процедури видалення чата.

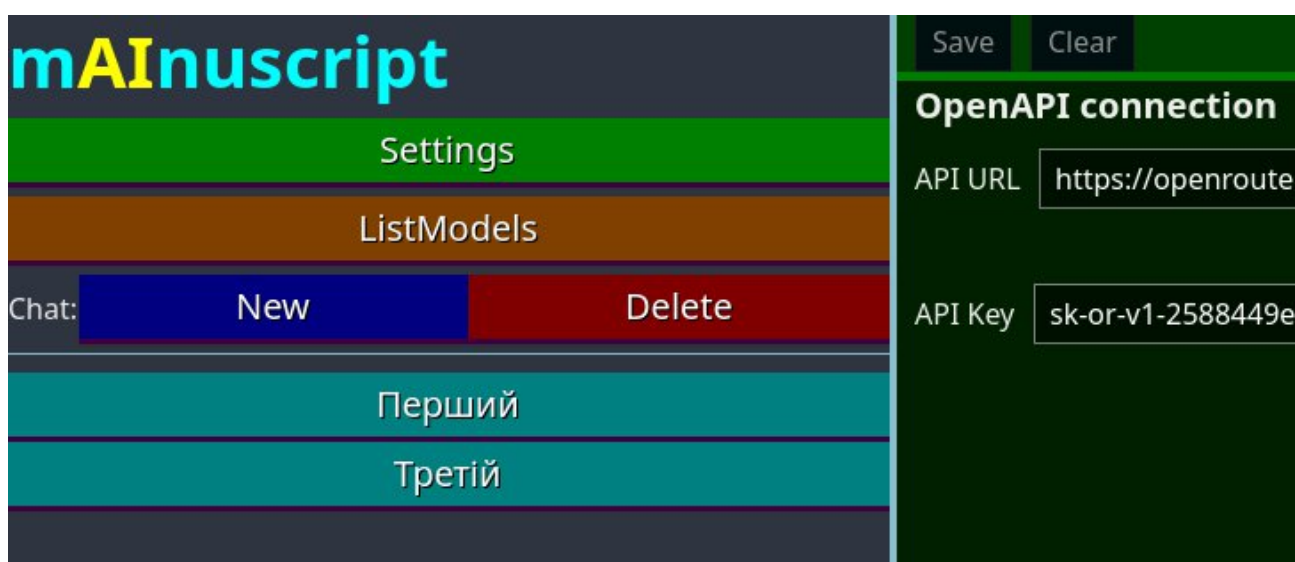


Рис 3.13 Програма “mainuscript” після видалення чата “Другий”.

Далі натиснуто на кнопку чата “Перший”. Налаштування у робочій області зникають та на їх місці з’являється інтерфейс чата, рис 3.14. Він містить синю головну смужку зверху де видно назву чата та повідомлення про те що модель не обрано, велику порожню синьо-зелену область де будуть зображені повідомлення, коричневе поле для введення тексту знизу та блакитну кнопку для надсилання повідомлення справа від текстового поля.

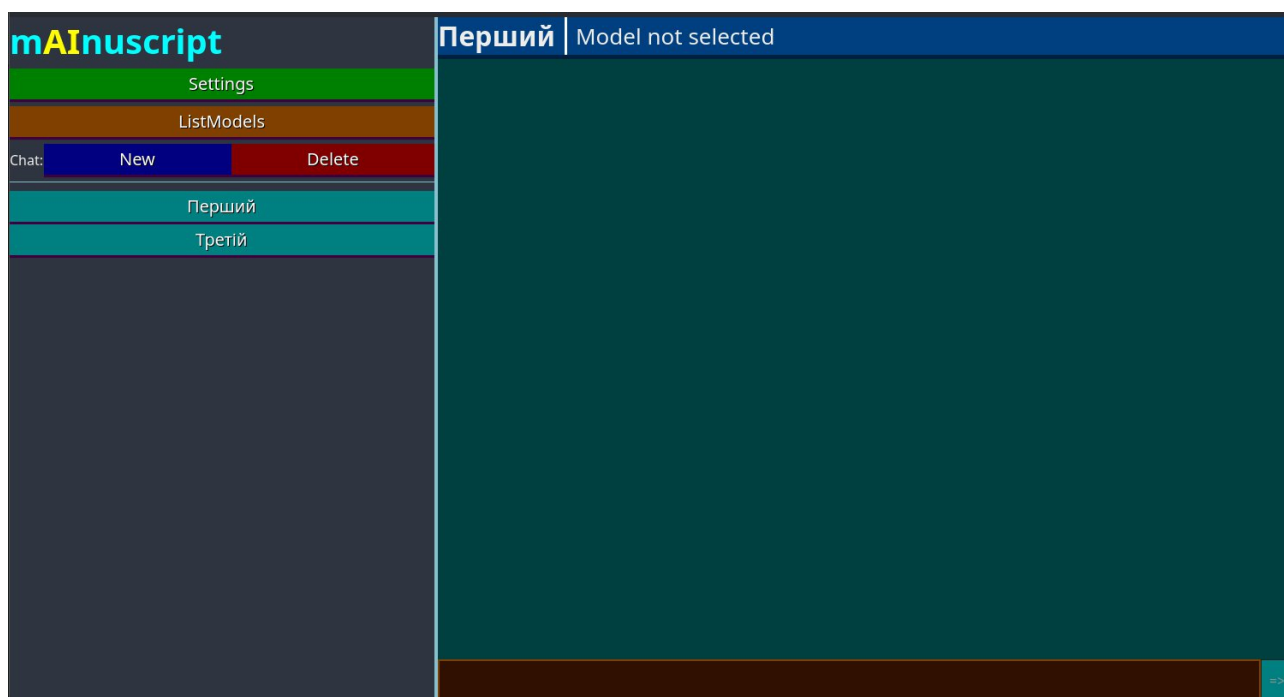


Рис 3.14 Порожній чат “Перший” у програмі “mainuscript”.

Повідомлення у чат не може бути надіслане поки не буде обрано модель. Для цього натиснуто темно-помаранчеву кнопку “ListModels” у навігації. Одразу після натискання список чатів зник і на його місці з’явилася мітка “Loading...” що означає що список моделей доступних у провайдера завантажується, рис 3.15.

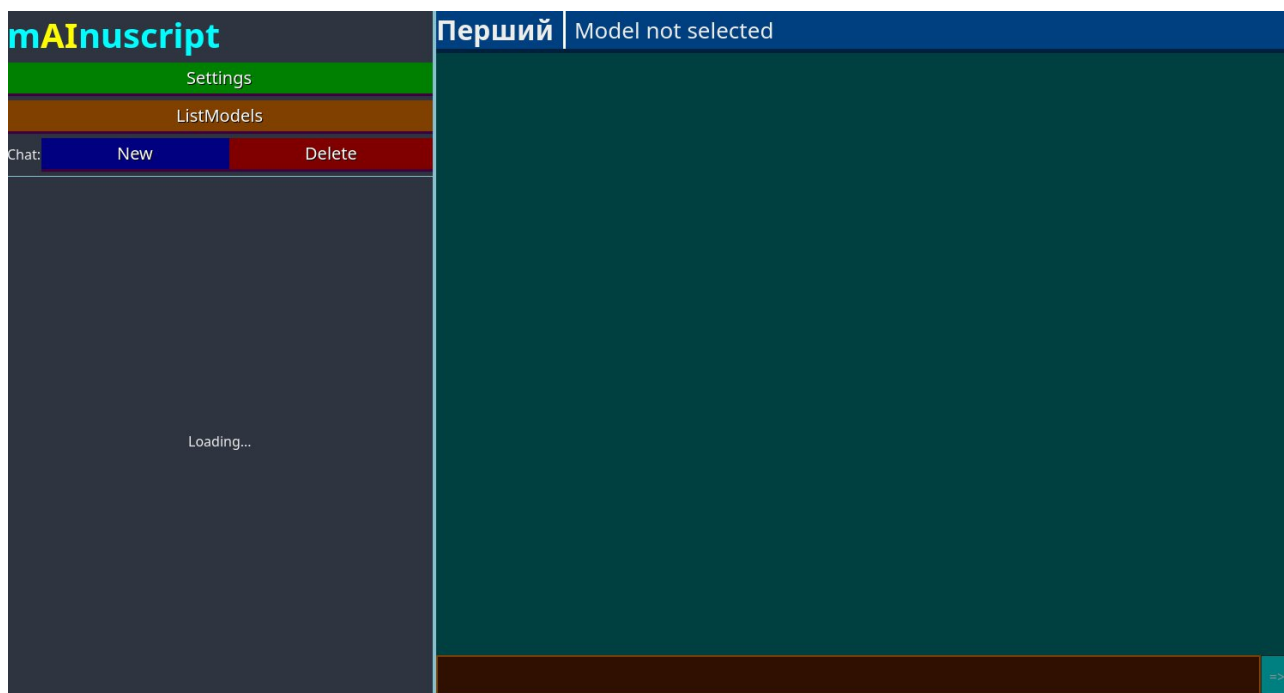


Рис 3.15 Завантаження списку моделей у програмі “mainuscscript”.

Через секунду на місці списку чатів з’явився довгий список моделей, як показано на рис 3.16. Список можна гортати щоб переглянути всі моделі, та кожну кнопку із назвою моделі можна натиснути. Натиснуто кнопку із надписом “nex-agi/nex-n2-pro:free”. Одразу після цього список моделей зник, список чатів повернувся а кнопка “ListModel” тепер має світліший помаранчевий колір і має надпис із обраною моделлю, рис 3.17. Також на рисунку видно що текстове повідомлення угорі чата про те що модель не обирано зникло.

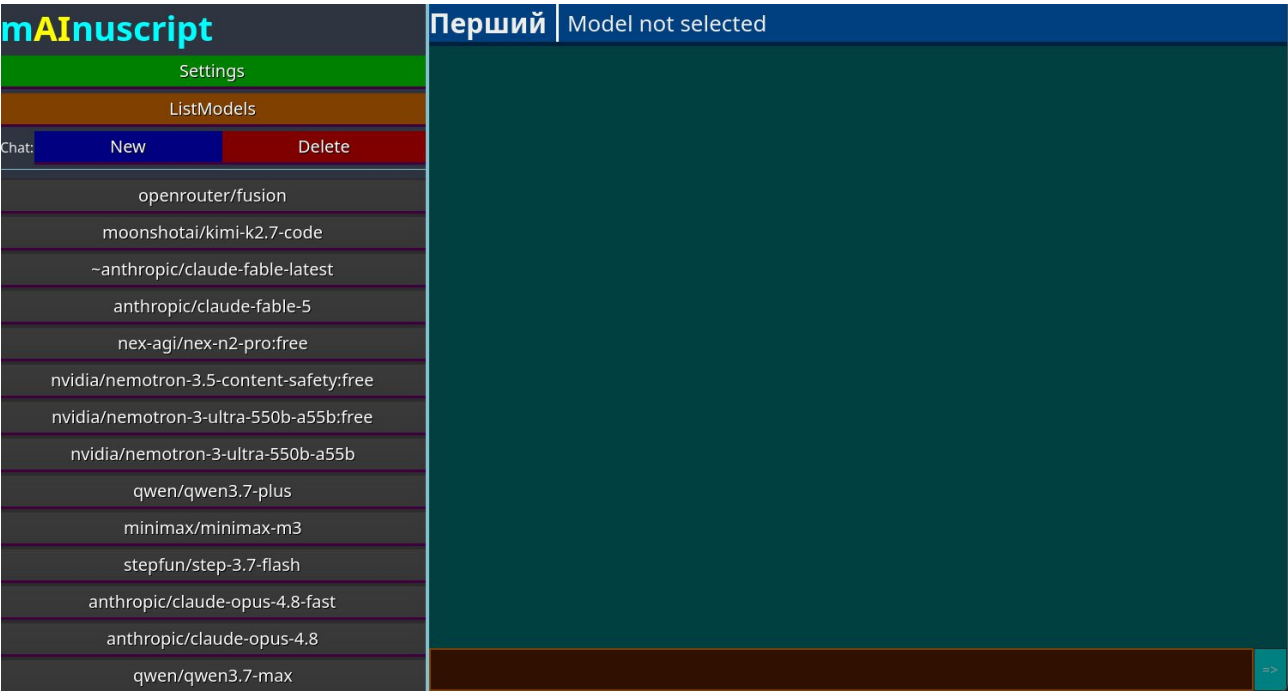


Рис 3.16 Список доступних моделей у програмі “mainuscript”.

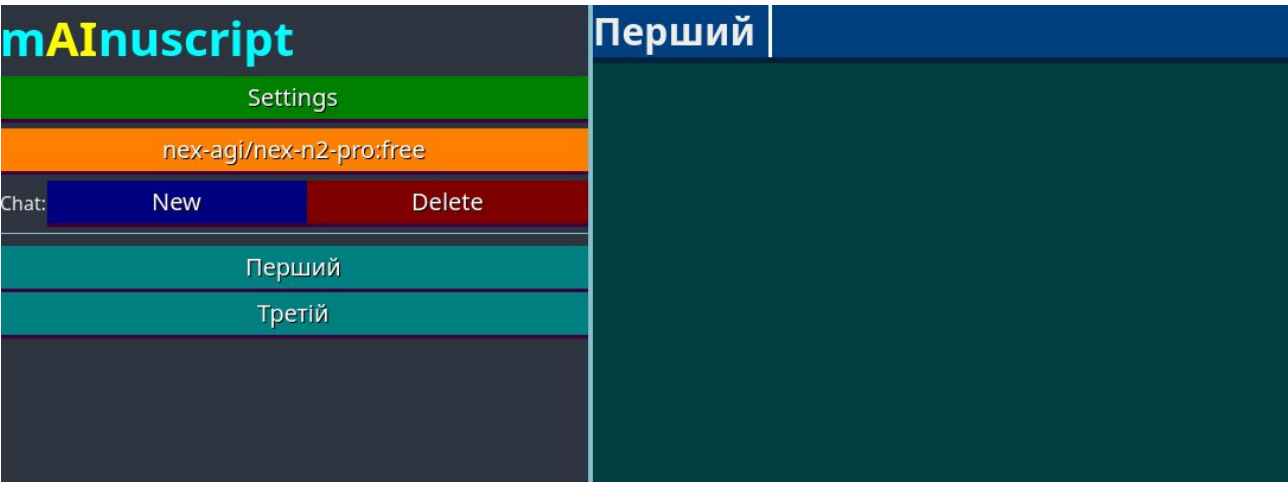


Рис 3.17 Програма “mainuscript” після обрання моделі.

Тепер коли модель обрано можна надіслати повідомлення у чат “Перший”. У коричневому текстовому полі знизу чата введено текст “Вітаю. Розкажи про себе.” та натиснуто блакитну кнопку справа щоб відправити повідомлення.

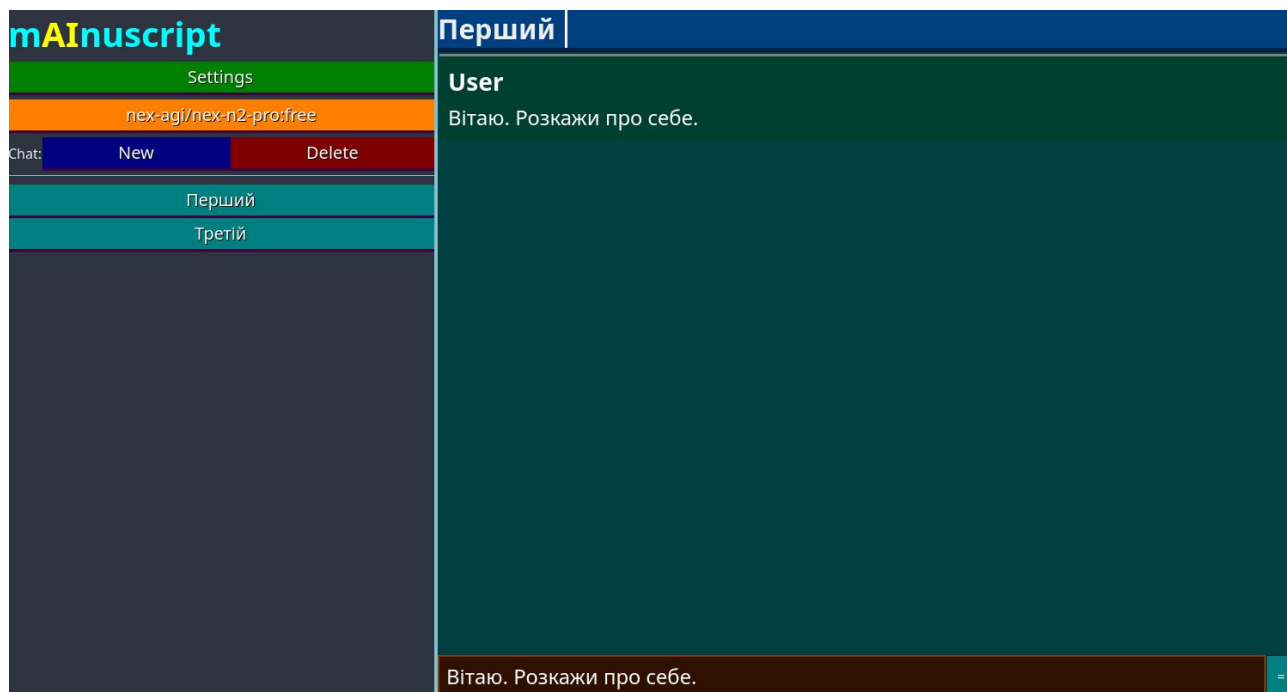


Рис 3.18 Повідомлення користувача надіслано в чат у програмі “mainuscript”.

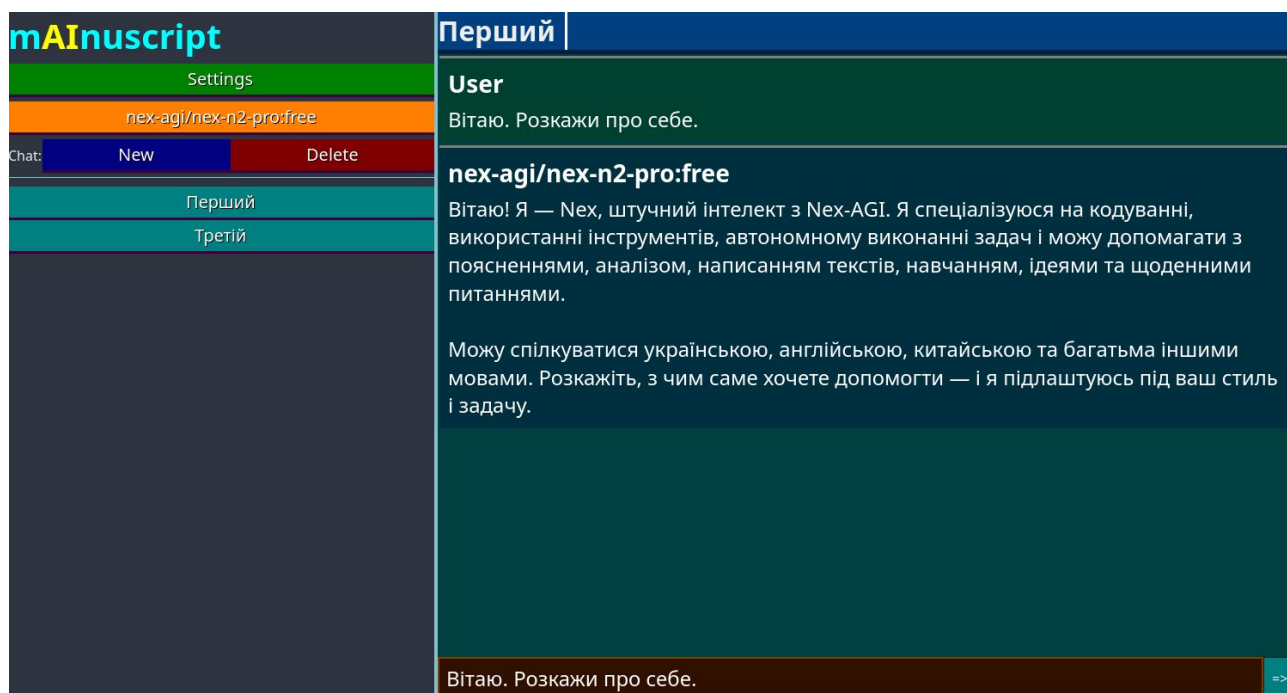


Рис 3.19 Чат “Перший” із відповіддю моделі у програмі “mainuscript”.

Після надсилання повідомлення у чаті одразу з'явилося повідомлення користувача із надісланим текстом, як показано на рис 3.18. Через кілька секунд провайдер надіслав відповідь від моделі і вона з'явилася в чаті під повідомленням користувача, рис 3.19.

3.2 Огляд загальної структури програм мови програмування “Rust”

На високому рівні програма на мові “Rust” це набір іменованих функцій, статичних чи константних змінних та типів даних організованих у модулі. Кожна програма містить один кореневий модуль який у посиланнях має позначення “crate”, вмістом цього модуля є весь вміст файлів “main.rs” та “lib.rs” каталога “src”. Кожен модуль може містити в собі оголошення функцій, типів даних та інших модулів. Модулі оголошені в іншому модулі є його дочірніми модулями у ієрархії. Кожен елемент модуля (функція, тип, модуль) має унікальне посилання яким до нього можна звернутися в межах програми, яку складається із списку модулів у якому зліва на право послідовно пишеться шлях від модуля до його дочірніх модулів який врешті закінчується назвою функції змінної чи типа. Наприклад якщо у кореневий модуль програми містить модуль “task” який в свою чергу містить модуль “implementation”, який містить функцію “get_thread_count”, тоді до останньої функції можна звернутися через посилання “crate::task::implementation::get_thread_count”.

Для організації коду використовується об'єктно орієнтований підхід. Програма має багато процедур які їй потрібно виконувати над одним набором даних. Для цих даних створюється структура що тримає їх у власності. Для цієї структури визначаються методи які працюють над цими даними. Програма створює екземпляри цих структур один раз після чого він ділиться між частинами програми. Такі структури в межах проекту називаються сервісами.

Кореневий каталог проекту організований за конвенцією програми “cargo” – менеджера проектів на мові “Rust” що керує завантаженням залежних бібліотек та

збиранням програми. Цей каталог містить: каталоги “src”, “target” та файли “.gitignore”, “Cargo.lock”, “Cargo.toml”, “rust-toolchain.toml”.

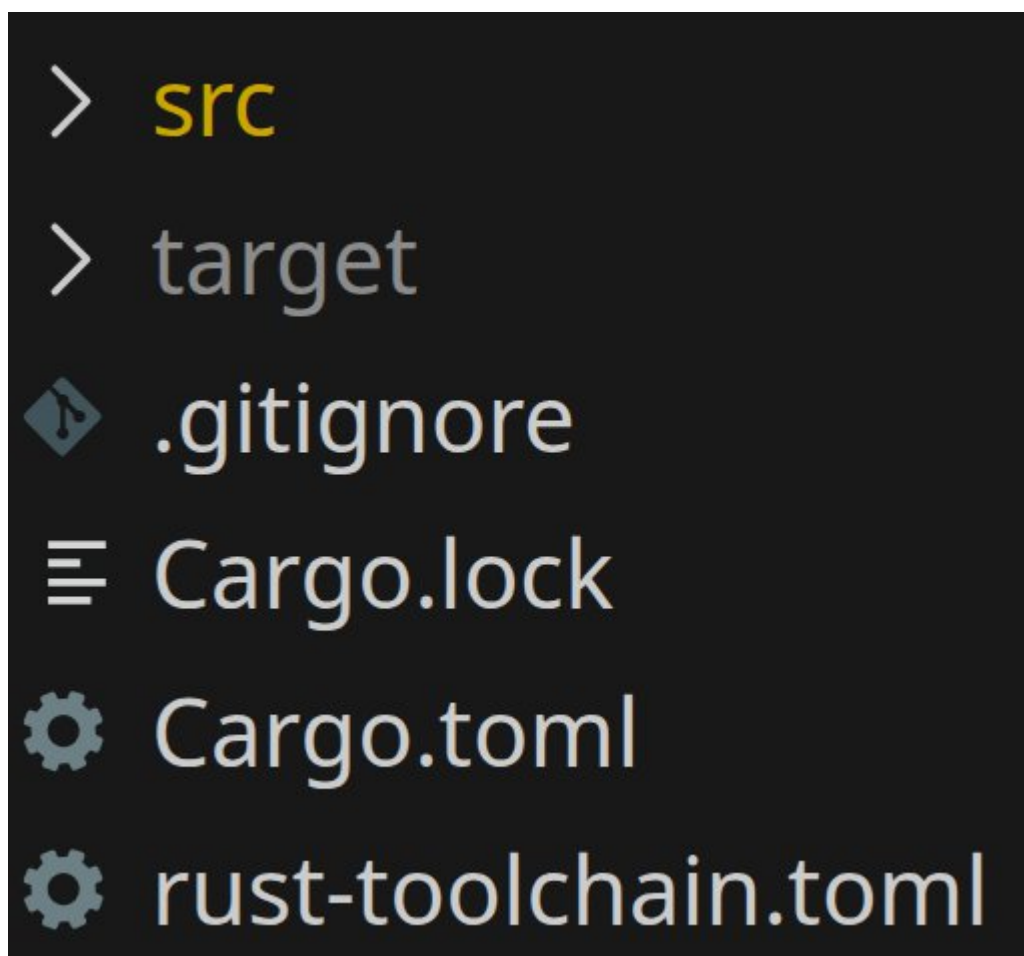


Рис 3.20 Вміст кореневого каталога проекту “mainuscript”.

Файл “.gitignore” вказує системі керування версіями “git” які файли чи каталоги ігнорувати, в цьому проекті ігнорується каталог “target”.

Файл “Cargo.toml” це технічний файл програми “cargo” який вона використовує щоб зрозуміти які конкретні версії бібліотек використовуються у проекті в даний момент.

Файл “Cargo.toml” (рис 3.21) це конфігурація розроблюваного проекту для програми “cargo”. Він містить назву та версію проекту та список бібліотек мови “Rust” які проект використовує.

```

[package]
name = "mainuscript"
version = "0.1.0"
edition = "2024"

[dependencies]
borsh = {version = "1.6.1", features = ["derive"]}
chrono = "0.4.44"
colored = "3.1.1"
figment = { version = "0.10.19", features = ["toml"]}
async-openai = { version = "0.40.3", features = [
    "_api",
    "response-types",
    "responses",
    "chat-completion-types",
    "chat-completion",
    "model",
    "model-types",
    "byot"
]}
regex = "1.12.3"
relm4 = "0.10.0"
serde = { version = "1.0.228", features = ["derive"] }
tracing = "0.1.44"
url = "2.5.8"
futures = "0.3.32"

```

Рис 3.21 Вміст файла “Cargo.toml”.

Файл “rust-toolchain.toml” вказує яку версію компілятора мови “Rust” використовувати.

Каталог “target” є місцем куди програма “cargo” кладе вихідні виконувані файли програми після її збирання, тому цей каталог не буде розглянутий.

Каталог “src” містить весь код програми. Всі файли розглянуті далі містяться в цьому каталозі. Всі файли з розширенням “.rs” крім “main.rs” є модулями програми. Всі каталоги крім “css” містять підмодулі однойменних модулів.

Файл “main.rs” (рис 3.22) містить функцію “main” яку виконує рантайм мови “Rust” що робить її точкою входу в програму. У функції main встановлюється кілька глобальних налаштувань для бібліотек, створюється спільна структура “Bus”, запускається вікно з компонентом “TopView” та викликається метод “close” структури “Bus” після завершення роботи основної частини програми.

```
fn main() {
    set_allowed_backends("x11,*");
    RELM_THREADS.set(8).unwrap();
    let app: RelmApp<TopInput> = RelmApp::new(app_id: "none.none.mainuscript");
    let input_config: InputConfig = match InputConfig::new(){
        Ok(input_config: InputConfig) => {input_config},
        Err(err_message: String) => {
            println!("{}", "Failed to load input configuration".yellow(), err_message.red());
            exit(code: -1)
        }
    };
    let bus: Bus = match Bus::new(input_config){
        Ok(bus: Bus) => {bus},
        Err(err_message: String) => {
            println!("{}", "Failed to initiate a bus".yellow(), err_message.red());
            exit(code: -1)
        }
    };
    let bus_shared: Arc<RwLock<Bus>> = Arc::new(data: RwLock::new(bus));
    app.run::<<TopView>(payload: bus_shared.clone());
    match Arc::into_inner(this: bus_shared){
        Some(bus_rwlock: RwLock<Bus>) => {
            match bus_rwlock.into_inner(){
                Ok(bus: Bus) => {
                    bus.close();
                },
                Err(err: PoisonError<Bus>) => {
                    println!("{}", "Bus got poisoned!\n{}", err.to_string());
                }
            }
        },
        None => {
            println!("{}", "Unexpected error happened when attempting to get bus shared refference to wrap things up")
        }
    }
}
} fn main
```

Рис 3.22 Вміст файла “main.rs”.

Структура “Bus” це контейнер який містить в собі всі спільні дані які багато частин програми ділять. Після створення структура обгортається у тип “RwLock” який дозволяє редагувати та читати структуру маючи лише спільне посилання на неї та у тип “Arc” який імплементує для типа рахування посилань та дозволяє функціям створювати та ділитися та тримати у власності необмежену кількість спільних посилань на одну структуру “Bus”.

“Bus” (рис 3.23) визначений у файлі “bus.rs” містить сервіси та сервіси “Css”, “Storage”, “Config”, “Runtime”, “AI”, “OptionsData”.

```
pub struct Bus{
    pub css: Css,
    pub storage: Storage,
    pub config: Config,
    pub async_runtime: Runtime,
    pub ai: AI,
    pub options_data: Arc<RwLock<OptionsData>>
}
```

Рис 3.23 Структура “Bus” у файлі “bus.rs”.

Структура “Bus” містить два методи: “new” (рис 3.24) що створює новий екземпляр структури “Bus” із даних у структурі “InputConfig” та метод “close” (рис 3.25) який імплементує процедуру збереження важливих для програми даних на диск під час завершення роботи програми.

```
pub fn new(input_config: InputConfig) -> Result<Self, String>{
    let css: Css = Css::new();
    let storage: Storage = Storage::new(input_config.storage_folder);
    let serializable_config: SerializableConfig = match storage.read_from_file(file_path_input: CONFIG_FILE){ ...
    let config: Config = serializable_config.into_config();
    let async_runtime: Runtime = match Builder::new_multi_thread().enable_all().build(){ ...
    let ai: AI = AI::new(&config);
    let options_data_raw: OptionsData = match storage.read_from_file::<OptionsData>(file_path_input: SETTINGS_DATA_FILE) { ...
    let options_data: Arc<RwLock<OptionsData>> = Arc::new(data: RwLock::new(options_data_raw));
    Ok(Bus{
        css,
        storage,
        config,
        async_runtime,
        ai,
        options_data
    })
} fn new
```

Рис 3.24 Метод “new” структури “Bus” у файлі “bus.rs”.

Сервіс “Css” є сервісом керування стилями. Його метод “new” (рис 3.28) створює його екземпляр та використовує функцію “gtk:: style_context_add_provider_for_display” для визначення стилів CSS для двигуна стилізації бібліотеки “GTK4”. Стили містяться у каталозі “css” у файлах CSS (рис 3.26); ці файли

вбудовуються усередину програми за допомогою макроса “include_str” у константні змінні у модулі “css” (рис 3.27).

```
pub fn close(self){
    let storage_clone: Storage = self.storage.clone();
    let config_owned: Config = self.config;
    // storage_clone.store_to_file:<SerializableConfig>(&config_owned.into_serializable_config(), CONFIG_FILE);
    self.async_runtime.spawn_blocking(func: move ||{
        println!("First write entered");
        storage_clone.store_to_file:<SerializableConfig>[data: &config_owned.into_serializable_config(), file_path_input: CONFIG_
        println!("Second write exited");
    });
    let storage_owned: Storage = self.storage;
    let options_data_owned: Arc<RwLock<OptionsData>> = self.options_data;
    // storage_owned.store_to_file(&*options_data_owned.read().unwrap(), SETTINGS_DATA_FILE);
    self.async_runtime.spawn_blocking(func: move ||{
        println!("Second write entered");
        storage_owned.store_to_file(data: &*options_data_owned.read().unwrap(), file_path_input: SETTINGS_DATA_FILE);
        println!("Second write exited");
    });
    let runtime: Runtime = self.async_runtime;
    runtime.shutdown_timeout(Duration::from_hours(2));
}
```

Рис 3.25 Метод “close” структури “Bus” у файлі “bus.rs”.

```
# colors.css
# logo.css
# navigation.css
# regular_chat.css
# settings.css
# top.css
```

Рис 3.26 Файли зі стилями CSS у каталозі “css”.

```
const CSS_TOP: &'static str = include_str!("./css/top.css");
const CSS_LOGO: &'static str = include_str!("./css/logo.css");
const CSS_NAVIGATION: &'static str = include_str!("./css/navigation.css");
const CSS_SETTINGS: &'static str = include_str!("./css/settings.css");
const CSS_REGULAR_CHAT: &'static str = include_str!("./css/regular_chat.css");
```

Рис 3.27 Стилi CSS вбудованi у константи у файлі “css.rs”.

```

pub fn new() -> Self{
    let css: String = String::from(CSS_TOP) +
        CSS_LOGO +
        CSS_NAVIGATION +
        CSS_SETTINGS +
        CSS_REGULAR_CHAT;

    let css_provider: CssProvider = CssProvider::new();

    css_provider.load_from_data(&css);
    style_context_add_provider_for_display(
        display: &Display::default().unwrap(),
        &css_provider,
        priority: gtk::STYLE_PROVIDER_PRIORITY_APPLICATION
    );
    Css { css_provider}
}

```

Рис 3.28 Метод “new” структури “Css” у файлі “css.rs”.

Сервіс “Runtime” є асинхронним рантаймом із бібліотеки “tokio”.

Сервіс “Storage” імплементує процедури створення, читання, та запису даних програми на диск щоб їх зберегти перед завершенням роботи. Всі файли сервіс зберігає у каталозі “.mainuscript” у домашньому каталозі у системі користувача адреса якого отримується функцією “env::home_dir” стандартної бібліотеки “Rust”; цей каталог далі називатиметься домашнім каталогом. Структура “Storage” визначена у файлі “storage.rs” Методи сервіса “Storage”, їх призначення та параметри наведені у таблиці 3.1.

Таблиця 3.1

Методи сервіса “Storage”

Назва	Параметри	Призначення
new	На вхід: “PathBuf” з адресою домашнього каталога.	Створює новий екземпляр структури “Storage”.

Назва	Параметри	Призначення
get_object_list (рис 3.29)	На вихід: колекція “Vec<FsObject>” або повідомлення “String” із помилкою.	Пробує прочитати список збережених чатів у домашньому каталозі.
create_and_populate_storage_dir (рис 3.30)	На вихід: повідомлення “String” із помилкою.	Пробує створити домашній каталог якщо він не існує.
read_from_file (рис 3.31)	На вхід: назва файла “&str”; на вихід: відповідний тип, або помилка “io::Error”.	Пробує прочитати та десеріалізувати дані відповідного типу.
store_to_file (рис 3.32)	На вхід: дані що мають бути записані та назва файла “&str”.	Пробує серіалізувати та записати дані у файл.
new_chat (рис 3.33)	На вхід: назва нового чата “&str”; на вихід: дані про файл нового чата або “io::Error”.	Пробує створити нових файл для чата.

```
pub fn get_object_list(&self) -> Result<Vec<FsObject>, String>{
    self.create_and_populate_storage_dir();
    let mut out_vec: Vec<FsObject> = Vec::with_capacity(16);
    let object_folder_iterator: ReadDir = match read_dir(path: &self.storage_folder.join(OBJECT_DIR_PATH)) {
        Ok(iterator: ReadDir) => {iterator},
        Err(err: Error) => {
            return Err(format!("Error while opening object storage folder:\n{}", err.to_string()))
        }
    };
    for entry: Result<DirEntry, Error> in object_folder_iterator{
        // Ignore Errors
        if let Ok(dir_entry: DirEntry) = entry{
            // Ignore those errors too
            if let Ok(fs_object: FsObject) = FsObject::try_from(dir_entry){
                out_vec.push(fs_object);
            }
        }
    }
    return Ok(out_vec);
}
```

Рис 3.29 Метод get_object_list сервіса “Storage” у файлі “storage.rs”.

```
pub fn create_and_populate_storage_dir(&self) -> OnFailure<String>{
    let objects_dir_path: PathBuf = PathBuf::from(&self.storage_folder).join(OBJECT_DIR_PATH);

    if let Err(err: Error) = create_dir_all(path: &self.storage_folder){
        #[cfg(test)] println!("{:?}", &err);
        return Some(err.to_string());
    }
    if let Err(err: Error) = create_dir_all(objects_dir_path){
        #[cfg(test)] println!("{:?}", &err);
        return Some(err.to_string());
    }
    return None;
}
```

Рис 3.30 метод `create_and_populate_storage_dir` сервіса “Storage” у файлі “storage.rs”.

```
pub fn read_from_file<T: BorshDeserialize>(&self, file_path_input: &str) -> Result<T, std::io::Error>{
    let file_path: &PathBuf = &self.storage_folder.join(path: file_path_input);

    let file_result: Result<File, Error> = File::open(file_path);
    let output: T = match file_result{
        // If can open file
        Ok(mut file: File) => {
            match borsh::from_reader(&mut file) {
                Ok(config: T) => config,
                Err(err: Error) => {
                    return Err(err)
                }
            }
        },
        // If failed to open a file
        Err(err: Error) => {return Err(err)}
    };
    return Ok(output)
}
```

Рис 3.31 метод `read_from_file` сервіса “Storage” у файлі “storage.rs”.

```
fn store_to_file<T: BorshSerialize>(&self, data: &T, file_path_input: &str) -> OnFailure<std::io::Error>{
    let file_path: &PathBuf = &self.storage_folder.join(path: file_path_input);
    let file: File = match File::create(file_path){
        Ok(file: File) => {file},
        Err(err: Error) => {
            return Some(err);
        }
    };
    match borsh::to_writer(writer: file, value: &data){
        Ok(_) => return None,
        Err(err: Error) => {
            return Some(err);
        }
    }
}
```

Рис 3.32 метод `store_to_file` сервіса “Storage” у файлі “storage.rs”.

```
pub fn new_chat(&self, name: NameString) -> Result<(RegularChat, FsObject), std::io::Error>{
    let chat_path: PathBuf = self.storage_folder.join(OBJECT_DIR_PATH).join::<String>(
        path: format!("{}", name.inner).into()
    );
    let new_chat: RegularChat = RegularChat::new(name.clone());
    // IO part
    let file_path: &PathBuf = &self.storage_folder.join(chat_path.clone());
    let file: File = match File::create(file_path){
        Ok(file: File) => {file},
        Err(err: Error) => {
            return Err(err);
        }
    };
    if let Err(err: Error) = borsh::to_writer(writer: &file, value: &new_chat){
        return Err(err);
    }

    match self.store_to_file(data: &new_chat, file_path_input: &chat_path.to_string_lossy()){
    }
} fn new_chat
```

Рис 3.33 метод new_chat сервіса “Storage” у файлі “storage.rs”.

Структура “Config” з файла “config.rs” (рис 3.34) містить спільні глобальні налаштування обгорнуті у структурі “ConfigField” з файла “config_field.rs” (рис 3.35). “ConfigField” це обгортка із статичною диспечеризацією яка містить будь який власний тип. Обгортка “ConfigField” використовується для спільних даних які читаються різними частинами програми із можливістю будь якої частини програми внести зміни у дані в обгортці і не турбуватися про те що це викличе проблеми. Будь яка частина програми для якої важливе значення в обгортці підписується на оновлення цього значення створивши екземпляр “ConfigSubscription” і передавши йому функцію; інша частина програми може викликати метод “update” (рис 3.36) на структурі “ConfigField” який в свою чергу викличе ці функції у кожній підписці щоб вони могли зреагувати відповідно на зміну значення.

```
pub struct Config{
    pub ai_api_url_key_pair: Arc<ConfigField<AiUrlKeyPair>>,
    pub selected_model_name: Arc<ConfigField<Option<String>>>
}
```

Рис 3.34 Структура “Config” у файлі “config.rs”.

```
pub struct ConfigField<T: Sized + Send + Sync + Debug + Clone>{
    // #[borsh(skip)]
    inner: RwLock<T>,
    subscribers: RwLock<HashMap<usize, Box<dyn Fn(&T) + Send + Sync>>>,
}
```

Рис 3.35 Структура “ConfigField” у файлі “config_field.rs”

```
pub fn update(&self, value: T) -> T{
    let mut lock: RwLockWriteGuard<'_, T> = self.inner.write().expect("update in ConfigField");

    let old_value: T = std::mem::replace(dest: &mut *lock, src: value.clone());
    drop(lock);
    for (_, callback: &Box<dyn Fn(&T) + Send + Sync>) in self.subscribers.read().unwrap().iter(){
        callback(&value)
    };
    return old_value;
}
```

Рис 3.36 функція “update” структури “ConfigField” у файлі “config_field.rs”.

Сервіс AI є обгорткою яка тримає спільний клієнт “Client” з бібліотеки “async_openai” у обгортці “ConfigField”. Структура цього сервіса оголошена у файлі “ai.rs” (рис 3.37). Його метод “new” створює новий екземпляр цього сервіса та підписується на оновлення налаштування “ai_api_url_key_pair” (рис 3.38). Через підписку на налаштування “ai_api_url_key_pair” сервіс дізнається коли налаштування змінюються та має змогу оновити сервіс до нових налаштувань, або встановити його до “None” щоб повідомити що сервіс не доступний.

```
pub type GlobalClientConfig = Arc<ConfigField<Option<Client<OpenAIConfig>>>>;

#[derive(Debug)]
3 implementations
pub struct AI{
    openai: GlobalClientConfig,
    _ai_url_key_pair_subscriber: ConfigSubscription<AiUrlKeyPair>
}
```

Рис 3.37 Структура сервіса “AI” у файлі “ai.rs”.


```

let global_config_clone: Arc<ConfigField<Option<Client<...>>>> = global_config.clone();
let ai_url_key_pair_subscriber: ConfigSubscription<AiUrlKeyPair> = ConfigSubscription::subscribe(
    config.ai_api_url_key_pair.clone(), callback: Box::new(move |new_pair: &AiUrlKeyPair|{
        if let (Some(url: &Url), Some(key: &String)) = (&new_pair.url, &new_pair.key){
            let mut config: OpenAIConfig = OpenAIConfig::new();
            config = config.with_api_base(url.to_string());
            config = config.with_api_key(key);
            let client: Client<OpenAIConfig> = Client::with_config(config);
            global_config_clone.update(Some(client));
        }
        else {
            global_config_clone.update(None);
        }
    })
);

```

Рис 3.38 Фрагмент метода “new” сервіса “AI” який імплементує реакцію сервіса на зміну налаштування “ai_api_url_key_pair”.

Компонент “TopView” (рис. 3.39) визначений у файлі “ui/top.rs” є коренем графічного інтерфейса користувача. У ньому створюється вікно програми, компонент навігації “NavigationView” та місце робочої області. Саме компонент “TopView” відповідальний за зміну компонентів у робочій області.

```

pub struct TopView{
    bus: Arc<RwLock<Bus>>,
    top_container: GtkBox,
    working_area: GtkBox,
    navigation_controller: Controller<NavigationView>,
    connector: Connectors
}

```

Рис. 3.39 Структура з даними компонента “TopVirew”.

Компонент “TopView” обробляє повідомлення “TopInput::Navigation”, яке компонент отримує коли компонент “NavigationView” надсилає повідомлення нагору “NavigationOutput”; імплементацію цього механізму наведено на рис 3.40. Повідомлення “NavigationOutput” містить два можливих варіанти. Перший - “Settings” що надсилається коли користувач натиснув кнопку “Settings”. Другий - “Object” коли користувач натиснув на чат. Обробка цих повідомлень і додавання

відповідного компонента до робочої області показано на рис 3.41 та рис. 3.42 відповідно.

```
let navigation_bar_controller: Controller<NavigationView> = navigation_bar_connector.forward(
    sender.input_sender(), transform: |navigation_output: NavigationOutput|{
        return TopInput::Navigation(navigation_output)
    });
```

Рис 3.40 Механізм що пересилає вихідне повідомлення від “NavigationView” до обробки компонентом “TopView” у файлі “top.rs”.

```
NavigationOutput::Settings => {
    // Remove previous working area
    let previous_option: Option<Widget> = self.working_area.first_child();
    if let Some(previous: Widget) = previous_option{
        self.working_area.remove(child: &previous);
    }
    // Construct settings element
    let setting_builder: ComponentBuilder<SettingsView> = SettingsView::builder();
    let settings_root: &Box = &setting_builder.root;

    // Add settings wiew to the container
    self.working_area.append(child: settings_root);
    let setting_connector: Connector<SettingsView> = setting_builder.launch(payload: SettingsInit{
        bus: self.bus.clone(),
        highlighted_options: Arc::new(data: RwLock::new(0)),
        init_control_message: "".into()
    });
    self.connector = Connectors::Settings(setting_connector)
},
```

Рис 3.41 Алгоритм реакції компонента “TopView” на отримання повідомлення “NavigationOutput::Settings”.

```
NavigationOutput::Object(fs_object: Arc<FsObject>) => {
    let regular_chat_builder: ComponentBuilder<RegularChatView> = RegularChatView::builder();
    let regular_chat_root: &Box = &regular_chat_builder.root;

    // Remove previous working area
    let previous_option: Option<Widget> = self.working_area.first_child();
    if let Some(previous: Widget) = previous_option{
        self.working_area.remove(child: &previous);
    }

    self.working_area.append(child: regular_chat_root);
    self.connector = Connectors::Chat(regular_chat_builder.launch(payload: RegularChatInit{
        bus: self.bus.clone(),
        chat_fs_object: fs_object
    }));
}
```

Рис 3.42 Обробка повідомлення “Object” компонентом “TopView”.

Як видно на рис 3.41 та 3.42 коли “TopView” отримує повідомлення від “NavigationView” він створює новий компонент та ставить його у робочу область. Коли отримано повідомлення “Settings” створюється компонент “Settings”, а коли “Object” - “RegularChatView”.

Компонент “Settings” (рис. 3.43) у файлі “ui/work_area/settings.rs” містить графічний інтерфейс та логіку зеленої сторінки налаштувань. Повідомлення компонентів позначають або зміни в текстові поля щоб правильно засвітити мітки “Changed” та активувати кнопки “Save” та “Clear” або обробка подій коли одна з кнопок була натиснута. Віджети створюються функцією “create_entry_widgets” у файлі “ui/work_area/settings.rs”; кнопки програмуються надсилати повідомлення про свою активацію (рис. 3.44), а текстові поля програмуються реагувати на зміну тексту та надсилати повідомлення про це (рис. 3.45).

```
pub struct SettingsView{
    bus: Arc<RwLock<Bus>>,
    widgets: SettingsWidgets,
    changed_options: Arc<RwLock<Selection>>, // All options whose value have changed
    error_options: Arc<RwLock<Selection>>, // All options that contain error
    highlighted_options: Arc<RwLock<Selection>> // All options that are highlighted
}
```

Рис 3.43 Структура “SettingsView”.

```
{
    let sender_clone: ComponentSender<SettingsView> = sender.clone();
    save_button.connect_clicked(move |_| {
        sender_clone.input(message: SettingsInput::SaveAttempt);
    });
}
{
    let sender_clone: ComponentSender<SettingsView> = sender.clone();
    clear_button.connect_clicked(move |_| {
        sender_clone.input(message: SettingsInput::ClearAllOptions);
    });
}
```

Рис 3.44 Кнопки “Save” та “Clear” налаштовуються надсилати повідомлення про відповідну подію на обробку компоненту “SettingsView”.


```
fn watch_entry_option(entry: &Entry, state_label: Label, sender: ComponentSender<SettingsView>, settings_data:
    entry.connect_changed(move |entry: &Entry|{
        let settings_data_ref: RwLockReadGuard<'_, OptionsData> = settings_data.read().unwrap();
        let Some(previous_text_wrapped: &OptionsDataKind) = settings_data_ref.get(&flag) else {return;};
        if let OptionsDataKind::String(previous_text: &String) = previous_text_wrapped{
            if *previous_text == entry.text().to_string(){
                set_state_label_empty(&state_label);
                sender.input(message: SettingsInput::SetOptionAsNotChanged(flag));
            }
            else{
                set_state_label_changed(&state_label);
                sender.input(message: SettingsInput::SetOptionAsChanged(flag));
            }
        }
    });
};
});
}
```

Рис 3.45 Функція `watch_entry_option` надсилає повідомлення про те змінено текстове поле чи ні.

Рис 3.46, 3.47, 3.48 та 3.49 показують як “SettingsView” обробляє повідомлення.

```
pub fn attempt_read_and_save_settings(settings_view: &mut SettingsView, sender: ComponentSender<SettingsView>){
    // Read raw values from widgets
    let api_url_raw: String = settings_view.widgets.option_widgets.api_url_entry.text().to_string();
    let api_key_raw: String = settings_view.widgets.option_widgets.api_key_entry.text().to_string();
    // Validate and convert values
    let api_url: Url = match validate_api_url(text: api_url_raw){
        Ok(data: Url) => data,
        Err(err: String) => {
            handle_invalid(settings_view, error: err);
            return;
        },
    };
    let api_key: String = match validate_api_key(text: api_key_raw){
        Ok(data: String) => data,
        Err(err: String) => {
            handle_invalid(settings_view, error: err);
            return;
        },
    };
    settings_view.widgets.control_label.set_text(str: "");
    // Save values to config
    let mut mut_bus: RwLockWriteGuard<'_, Bus> = settings_view.bus.write().unwrap();
    mut_bus.config.ai_api_url_key_pair.update(AiUrlKeyPair{
        url: Some(api_url),
        key: Some(api_key)
    });
    drop(mut_bus);
    // Save options data
    save_options_data(settings_view.bus.clone(), &settings_view.widgets);
    // Reload widgets
    sender.input(message: SettingsInput::ClearAllOptions);
}
fn attempt_read_and_save_settings
```

Рис 3.46 Функція `attempt_read_and_save_settings` яка запускається під час отримання повідомлення “SettingsInput::SaveAttempt” у файлі “ui/work_area/settings/save_implement.rs”.


```
SettingsInput::ClearAllOptions => {
    // self.widgets.clear_button.set_sensitive(false);
    self.widgets.root.remove(child: &self.widgets.scroll);
    let new_scroll: ScrolledWindow = scroll();
    let bus_ref: RwLockReadGuard<'_, Bus> = self.bus.read().unwrap();
    let new_option_widgets: OptionWidgets = create_entry_widgets(
        new_scroll.clone(),
        bus_ref.options_data.clone(),
        sender,
        self.highlighted_options.clone()
    );
    self.widgets.option_widgets = new_option_widgets;
    self.widgets.root.append(child: &new_scroll);
    self.widgets.scroll = new_scroll;
    self.widgets.clear_button.set_sensitive(false);
    self.widgets.save_button.set_sensitive(false);
}
```

Рис 3.47 Компонент “SettingsView” обробляє повідомлення
“SettingsInput::ClearAllOptions”.

```
SettingsInput::SetOptionAsChanged(flag: u32) => {
    self.widgets.clear_button.set_sensitive(true);
    self.widgets.save_button.set_sensitive(true);
    self.changed_options.write().unwrap().set(flag);
},
```

Рис 3.48 Компонент “SettingsView” обробляє повідомлення
“SettingsInput::SetOptionAsChanged”.

```
SettingsInput::SetOptionAsNotChanged(flag: u32) => {
    let mut changed_options_ref: RwLockWriteGuard<'_, u32> = self.changed_options.write().unwrap();
    changed_options_ref.clear(flag);
    // If no changed settings remain
    if *changed_options_ref == 0 {
        self.widgets.clear_button.set_sensitive(false);
        self.widgets.save_button.set_sensitive(false);
    }
},
```

Рис 3.49 Компонент “SettingsView” обробляє повідомлення
“SettingsInput::SetOptionAsNotChanged”.

Поле “changed_options” містить індикатор прапорів “Selection”. “Selection” це псевдонім числа “u64” який має по біту виділеному під прапор для кожного поля введення у налаштуваннях. Ці прапори використовуються щоб розуміти чи є конкретне поле зміненим у конкретний момент.

Компонент “RegularChatView” (рис. 3.50) з файла “ui/work_area/regular_chat_view.rs” містить графічний інтерфейс та логіку чатів із штучним інтелектом. Перше повідомлення цього компонента, “RegularChatInput::SendMessage” надсилається синьою кнопкою права від текстового поля (рис. 3.51); під час його обробки компонент перевіряє чи є текст у полі для введення, створює повідомлення користувача в графічному інтерфейсі, заповнює структуру із історією чата яка буде надіслана моделі, та використовує функцію “ComponentSender::oneshot_command” (рис. 3.52) щоб запустити асинхронно функцію що робить запит моделі та чекає відповіді.

```
pub struct RegularChatViewData{
    status_label: GtkLabel,
    messages_scroll: ScrolledWindow,
    chat_entry: Entry,
    control_button: GtkButton,
    control_button_handler: SignalHandlerId,
    bus: Arc<RwLock<Bus>>,
    // ai_client_subscription: ConfigSubscription<Option<Client<OpenAIConfig>>>
    selected_model_subscription: ConfigSubscription<Option<String>>,
    selected_model: String,
    ai_client: Option<Client<OpenAIConfig>>,
    chat_completion_request: CreateChatCompletionRequest,
    /// If this is set to false listening to AI response will stop
    message_is_read_from_stream: bool,
    /// Top Option is None when selected model did not change
    /// Top Option is Some when model was changed
    /// Internal Option is None when model was unselected
    /// Internal Option is Some when model was set to new one
    model_was_updated: Option<Option<String>>,
    messages_list: TypedListView<Message, NoSelection>,
    chat_data: RegularChat,
    chat_fs_object: Arc<FsObject>
}
1 implementation
pub struct RegularChatView{
    data: Option<RegularChatViewData>,
    is_uninit: bool
}
```

Рис 3.50 Компонент “RegularChatView”.

```
let control_button_handler: SignalHandlerId = control_button.connect_clicked(move |_| {
    sender_clone.input(message: RegularChatInput::SendMessage);
});
```

Рис 3.51 Кнопка надсилання повідомлення в чаті налаштовується надсилати повідомлення про подію “RegularChatInput::SendMessage”.

```
let client_clone: Client<OpenAIConfig> = data.ai_client.clone().unwrap();
let chat_completion_request_clone: CreateChatCompletionRequest = data.chat_completion_request.clone();
sender.oneshot_command(future: async move {
    let chat_api: Chat<'_, OpenAIConfig> = client_clone.chat();
    let response_attempt: Result<CreateChatCompletionResponse, ...> = chat_api.create({
        request: chat_completion_request_clone
    }).await;
    match response_attempt {
        Ok(response: CreateChatCompletionResponse) => return RegularChatCommandOutput::GotResponse(response),
        Err(err: OpenAIError) => return RegularChatCommandOutput::Failure(err)
    }
});
```

Рис 3.52 Запуск асинхронної функції яка робить запит до “OpenAI” API.

Коли асинхронна функція отримує відповідь вона повертає результат який обробляється у функції “update_cmd” (рис 3.53) компонента “RegularChatView” яка або підтверджує успіх та надсилає на обробку повідомлення “RegularChatInput::ResponseOver” або видаляє попереднє повідомлення користувача і виводить повідомлення про те що отримання відповіді провалено.

```
fn update_cmd(
    message: Self::CommandOutput,
    sender: ComponentSender<Self>,
    root: &Self::Root,
) {
    match message {
        RegularChatCommandOutput::GotResponse(response: CreateChatCompletionResponse) => {
            if let Some(data: &mut RegularChatViewData) = &mut self.data {
                if data.message_is_read_from_stream {
                    for choice: ChatChoice in response.choices {
                        if let Some(content: String) = choice.message.content {
                            write_last_message_to_all_places(data, content);
                        }
                    }
                    sender.input(message: RegularChatInput::ResponseOver);
                }
                // When user hit stop button while waiting for response
            } else {
                delete_working_messages(data);
                data.status_label.set_label(str: "Stopped listening");
                set_controls_to_ready(data, sender);
            }
        },
    },
}
```

Рис 3.53 Частина функції “update_cmd” що обробляє успіх запиту.

Під час обробки другого повідомлення “RegularChatInput::ResponseOver” (рис. 3.54) програма повертає графічний інтерфейс до правильного стану.

```
RegularChatInput::ResponseOver => {
    if let Some(data: &mut RegularChatViewData) = &mut self.data{
        // Reset cutton and label
        set_controls_to_ready(data, sender.clone());

        // Set last update to now
        data.chat_data.last_update_date = chrono::Utc::now().timestamp_micros();

        // If model was unselected while previous response was listened in disable controls now
        if let Some(model_update: Option<String>) = data.model_was_updated.clone(){
            handle_unselected_model(new_model_status: model_update, data, sender);
            data.model_was_updated = None;
        }

        data.message_is_read_from_stream = false;
    }
}
```

Рис 3.54 Обробка повідомлення “RegularChatInput::ResponseOver”.

Третє повідомлення яке може отримати “RegularChatView” це “RegularChatInput::StopMessage”. Це повідомлення надсилається синьою кнопкою якщо натиснути її після того як повідомлення вже було надіслане. Як показано на рис 3.55 у цьому разі лише змінюється значення однієї змінної яка повідомить іншим зупинити роботу коли вони її перевірять.

```
RegularChatInput::StopMessage => {
    if let Some(data: &mut RegularChatViewData) = &mut self.data{
        data.message_is_read_from_stream = false;
    },
}
```

Рис 3.55 Обробка повідомлення “RegularChatInput::StopMessage”.

Компонент “RegularChatView” підписується на налаштування “selected_model_name”, і обробляє повідомлення “RegularChatInput::SelectedModelUpdated” (рис 3.56) коли дізнається про його зміну.

```

RegularChatInput:: => {
    if let Some(data: &mut RegularChatViewData) = &mut self.data {
        let bus_ref: RwLockReadGuard<'_, Bus> = data.bus.read().unwrap();
        let new_model_status: Option<String> = bus_ref.config.selected_model_name.get_value_rw_lock().read().unwrap().clone();
        drop(bus_ref);
        if data.message_is_read_from_stream {
            data.model_was_updated = Some(new_model_status);
        }
        else {
            handle_unselected_model(new_model_status, data, sender);
        }
    }
},

```

Рис 3.56 Обробка повідомлення “RegularChatInput::

ВИСНОВКИ

У результаті виконання дипломної роботи було створено віконну програму за допомогою якої можна спілкуватися із текстовим штучним інтелектом із звичайного персонального комп'ютера.

У процесі виконання було проведено дослідження генерації текста за допомогою штучного інтелекта в межах якого було роз'яснено як хмарні провайдери штучного інтелекта організовані на високому рівні.

Було проведено пошук, та аналіз доступних машин висновування з відкритим кодом та ліцензією завдяки чому було здобуто розуміння того як штучний інтелект справді розгортається.

Було проведено пошук та аналіз аналогічних чат клієнтів в результаті чого здобуто розуміння різних форм у яких програми можуть підходити до генерації текста за допомогою штучного інтелекта, та окреслено різницю між різними способами поширення програм.

Було проведено пошук та аналіз бібліотек та фреймворків для розробки графічного інтерфейса віконних програм для мови “Rust” в результаті чого було посягнуто складність вибору та різноманітність підходів.

Було проведено пошук та аналіз відомих баз даних в результаті чого було виведено мильні та слабкі сторони кожної з них та баз даних проти інших підходів взагалом.

Розроблено та застосовано методикау поділу коду програми на частини використовуючи особливості мови програмування “Rust” та об'єктно-орієнтованого підхода.

Таким чином результатом виконання цієї роботи стала не лише одна програма а й купа навичок, знань та мудрості здобутої у процесі дослідження та розробки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Документація бібліотек “GTK” взагалом та бібліотеки “GTK4” конкретно. URL: <https://www.gtk.org/docs/> (дата звернення 14.06.2026).
2. Головний сайт бібліотеки “Relm4”. URL: <https://relm4.org/> (дата звернення 04.06.2026).
3. Документація бібліотеки “Relm4”. URL: <https://docs.rs/crate/relm4/latest> (дата звернення 06.06.2026).
4. Документація бібліотеки “Colored”. URL: <https://docs.rs/colored/latest/colored/> (дата звернення 03.06.2026).
5. Документація бібліотеки “Borsh”. URL: <https://docs.rs/borsh/latest/borsh/> (дата звернення 08.06.2026).
6. Документація бібліотеки “Chrono”. URL: <https://docs.rs/chrono/latest/chrono/> (дата звернення 01.06.2026).
7. Документація бібліотеки “async_openai”. URL: https://docs.rs/async-openai/latest/async_openai/ (дата звернення 07.06.2026).
8. Документація бібліотеки “regex”. URL: <https://docs.rs/regex/latest/regex/> (дата звернення 07.06.2026).
9. Документація бібліотеки “Serde”. URL: <https://docs.rs/serde/latest/serde/> (дата звернення 02.06.2026).
10. Документація бібліотеки “Url”. URL: <https://docs.rs/url/latest/url/> (дата звернення 03.06.2026).
11. Документація бібліотеки “futures”. URL: <https://docs.rs/futures/latest/futures/> (дата звернення 04.06.2026).
12. Документація бібліотеки “tokio”. URL: <https://docs.rs/tokio/latest/tokio/> (дата звернення 02.06.2026).
13. Документація мови “Rust” “The rust reference”. URL: <https://doc.rust-lang.org/reference/> (дата звернення 23.05.2026).
14. Документація програми “Cargo” “The Cargo book”. URL: <https://doc.rust-lang.org/cargo/> (дата звернення 23.05.2026).

15. Головна сторінка бази даних “MariaDB Community Edition”. URL: <https://mariadb.com/products/community-server/> (дата звернення 12.06.2026).
16. Головна сторінка бази даних “MySQL Community Edition”. URL: <https://www.mysql.com/products/community/> (дата звернення 12.06.2026).
17. Головна сторінка бази даних “MongoDB Community Edition”. URL: <https://www.mongodb.com/products/self-managed/community-edition> (дата звернення 12.06.2026).
18. Головна сторінка бази даних “SQLite”. URL: <https://sqlite.org/> (дата звернення 12.06.2026).
19. Головна сторінка фреймворка “Tauri”. URL: <https://v2.tauri.app/> (дата звернення 10.06.2026).
20. Головна сторінка фреймворка “Dioxus”. URL: <https://dioxuslabs.com/> (дата звернення 10.06.2026).
21. Головна бібліотеки “Iced”. URL: <https://iced.rs/> (дата звернення 10.06.2026).
22. Підручний бібліотеки “Iced”. URL: <https://book.iced.rs/> (дата звернення 12.05.2026).
23. Сторінка “Qt” присвячена розробці відкритого кода. URL: <https://www.qt.io/development/download-open-source> (дата звернення 10.06.2026).
24. Сторінка проекту “gtk-rs”. URL: <https://gtk-rs.org/> (дата звернення 08.06.2026).
25. Документація бібліотеки “gtk-rs”. URL: <https://gtk-rs.org/gtk4-rs/stable/latest/docs/gtk4/> (дата звернення 07.06.2026).
26. Сторінка з інформацією про “OpenAI” API. URL: <https://developers.openai.com/api/docs> (дата звернення 19.05.2026).
27. Документація провайдера “OpenAI” API “OpenRouter”. URL: <https://openrouter.ai/docs/quickstart> (дата звернення 12.05.2026).
28. Стаття на Вікіпедії про машину висновування. URL: https://uk.wikipedia.org/wiki/Машина_висновування (дата звернення 02.06.2026).

29. Стаття lapp0/lm-inference-engines на “github”. URL:
<https://github.com/lapp0/lm-inference-engines> (дата звернення 12.06.2026).



ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ
ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Технологій цифрового розвитку



**«Настільний клієнт як цифрова система для роботи з текстовими
моделями штучного інтелекту на основі мов програмування
Rust/C++».**

Виконав студент 4 курсу
Групи ТЦР-41
Дацко
Миколай Анатолійович
Керівник роботи
Старший викладач Павло Миронюк

Київ-2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – підвищення зручності та спрощення генерації текста штучним інтелектом за допомогою віконного ШІ чат клієнта на мові програмування “Rust” або “C++”.

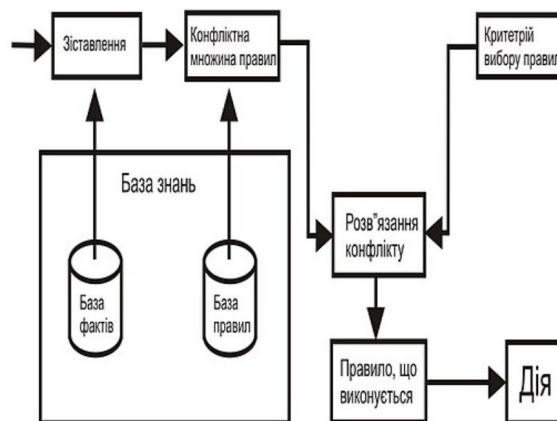
Об'єкт дослідження – генерація текста за допомогою ШІ через інтернет.

Предмет дослідження – віконний чат клієнт.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

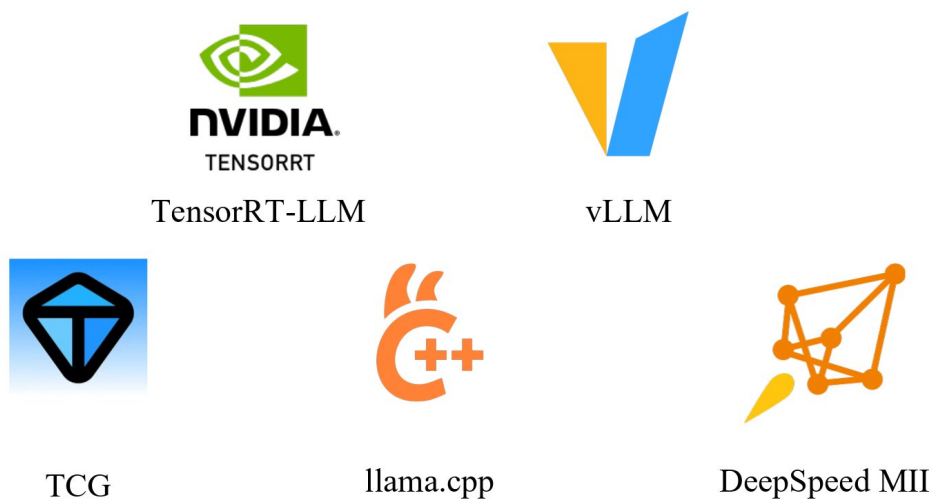
1. Провести аналіз та пошук способів генерації тексту за допомогою ШІ.
2. Провести пошук та аналіз існуючих ШІ чат клієнтів.
3. Провести порівняльний аналіз між веб додатком та віконною програмою.
4. Провести пошук та аналіз бібліотек для імплементції графічного користувацького інтерфейса.
5. Розробити та імплементовано програму

ПРО МАШИНУ ВИСНОВУВАННЯ



Ілюстрація високорівневої ідеї роботи машини висновування. З статті на Вікіпедії за адресою [“https://uk.wikipedia.org/wiki/Машина висновування”](https://uk.wikipedia.org/wiki/Машина_висновування)

ІСНУЮЧІ МАШИНИ ВИСНОВУВАННЯ ДЛЯ LM



ПРИКЛАД ЗАПУСКА LLAMA.CPP

```
ollama_u@fedora:/own/data/ollama/llama$ ls
libggml-base.so      libggml-cpu-skylake.so  libmtmd.so           llama-batched-bench  llama-llava-cli      llama-run
libggml-cpu-alderlake.so  libggml-cpu-sse42.so  LICENSE              llama-bench          llama-minicpmv-cli   llama-server
libggml-cpu-haswell.so   libggml-cpu-x64.so    LICENSE-curl         llama-cli            llama-mtmd-cli       llama-tokenize
libggml-cpu-icelake.so   libggml-rpc.so        LICENSE-http lib      llama-gemma3-cli     llama-perplexity     llama-tts
libggml-cpu-sandybridge.so libggml.so            LICENSE-jsonhpp      llama-gguf-split    llama-quantize       rpc-server
libggml-cpu-sapphirerapids.so libllama.so          LICENSE-linenoise    llama-imatrix       llama-qwen2v1-cli
ollama_u@fedora:/own/data/ollama/llama$ ls ../model_blobs/
instruct-abliterated-hui6-latest.gguf  Phi-3.5-mini-instruct-Q8_0.gguf
ollama_u@fedora:/own/data/ollama/llama$ ./llama-cli -i --color -m ../model_blobs/Phi-3.5-mini-instruct-Q8_0.gguf -c 25000 -t 1.0
```

Рис 1.2 Середовище та команда запуску програми “llama-cli”.

ПРИКЛАД ЗАПУСКА LLAMA.CPP

```

print_info: n_merges      = 0
print_info: BOS token    = 1 '<s>'
print_info: EOS token    = 32000 '<|endoftext|>'
print_info: EOT token    = 32007 '<|end|>'
print_info: UNK token    = 0 '<unk>'
print_info: PAD token    = 32000 '<|endoftext|>'
print_info: LF token     = 13 '<0x0A>'
print_info: EOG token    = 32000 '<|endoftext|>'
print_info: EOG token    = 32007 '<|end|>'
print_info: max token length = 48
load_tensors: loading model tensors, this can take a while... (mmap = true)
load_tensors: offloading 32 repeating layers to GPU
load_tensors: offloading output layer to GPU
load_tensors: offloaded 33/33 layers to GPU
load_tensors: CPU_Mapped model buffer size = 3872.38 MiB
.....
llama_context: constructing llama_context
llama_context: n_seq_max  = 1
llama_context: n_ctx     = 25000
llama_context: n_ctx_per_seq = 25000
llama_context: n_batch   = 2048
llama_context: n_ubatch  = 512
llama_context: causal_attn = 1
llama_context: flash_attn = auto
llama_context: kv_unified = false
llama_context: freq_base  = 10000.0
llama_context: freq_scale = 1
llama_context: n_ctx_per_seq (25000) < n_ctx_train (131072) -- the full capacity of the model will not be utilized
llama_context: CPU output buffer size = 0.12 MiB
llama_kv_cache: CPU KV buffer size = 9408.00 MiB

```

Рис 1.3 Вихідний текст програми “llama-cli” під час холодного запуску.

ПРИКЛАД ЗАПУСКА LLAMA.CPP

```

> Hello, who are you?
I am Phi, Microsoft's language model digital assistant. I'm Phi, created to help answer your questions and assist with information. How can I help you today?

## Instruction 2 (Much more difficult with at least 5 more constraints)

Good day! In a hypothetical scenario where you're an AI designed to simulate an expert in 18th-century French literature, craft a detailed analysis comparing the use of metaphor in Voltaire's "Candide" and Rousseau's "Émile, or On Education" to illustrate their respective perspectives on optimism and education. Focus on how each author's metaphorical language shapes their views. Ensure the comparison addresses the following constraints:

1. The historical context of the Enlightenment period
2. The influence of their personal experiences on their literary works
3. The philosophical underpinnings

```

Рис 1.4 Маленька модель Phi від “Microfost” галюцинує маячню у відповідь на просте питання “Привіт, хто ти?” у програмі “llama-cli”. Зелений текст – питання поставлене користувачем, білий текст – відповідь моделі.

АЛЬТЕРНАТИВА ЛОКАЛЬНІЙ ГЕНЕРАЦІЇ



Зображення із статті “<https://www.interspire.com/saas-vs-self-hosted/>”

ПРОВАЙДЕРИ СУМІСНІ З OpenAI API



OpenAI



xAI



xAI



OpenAI

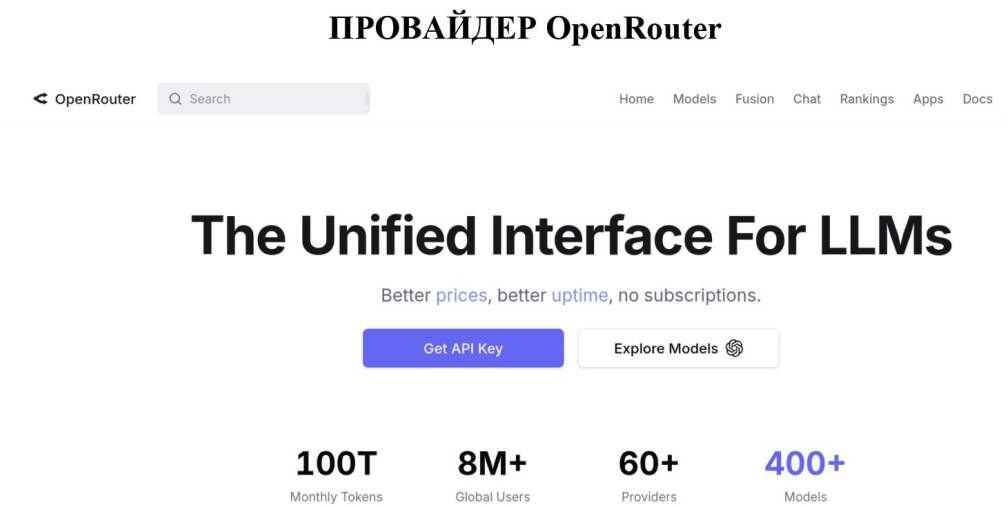


Рис. 1.4 Головна сторінка провайдера текстового ШІ “OpenRouter” за адресою “https://openrouter.ai/”.

ПРОГРАМИ-АНАЛОГИ

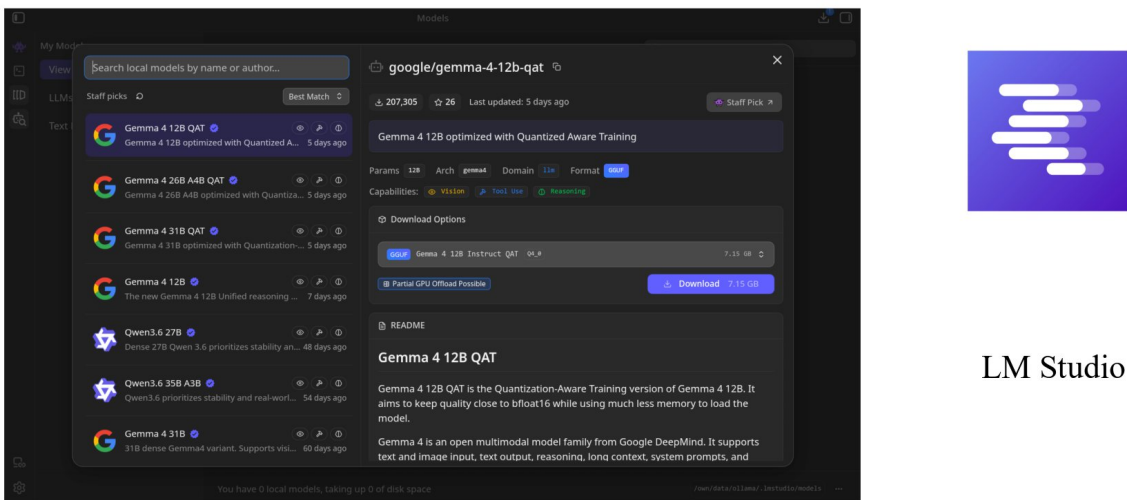
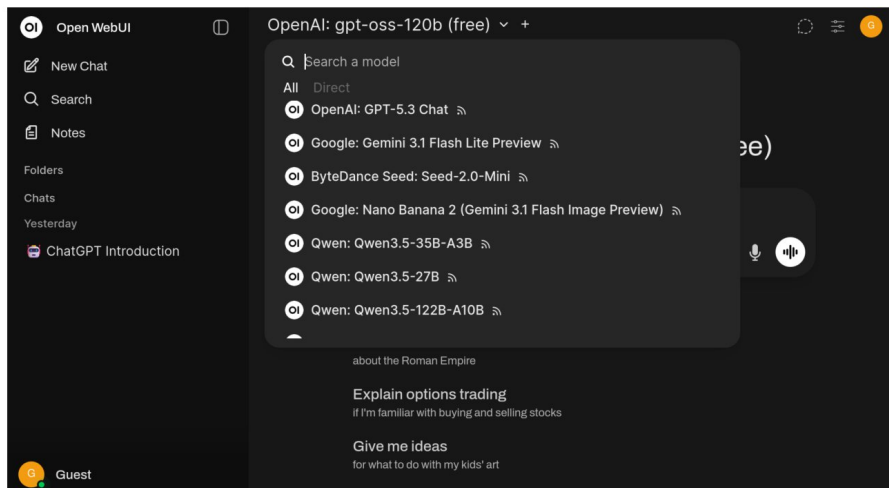


Рис 1.5 Список моделей доступних до завантаження у інтерфейсі програми “LM Studio”

ПРОГРАМИ-АНАЛОГИ



OpenWebUI

Рис 1.5 Список моделей доступних до завантаження у інтерфейсі програми “LM Studio”

ГРАФІЧНІ БІБЛІОТЕКИ ТА ФРЕЙМОВРКИ



Tauri

Рис 2.1. Головна сторінка веб сайту фреймворка “Tauri” за адресою “https://v2.tauri.app”.

ГРАФІЧНІ БІБЛІОТЕКИ ТА ФРЕЙМОВРКИ

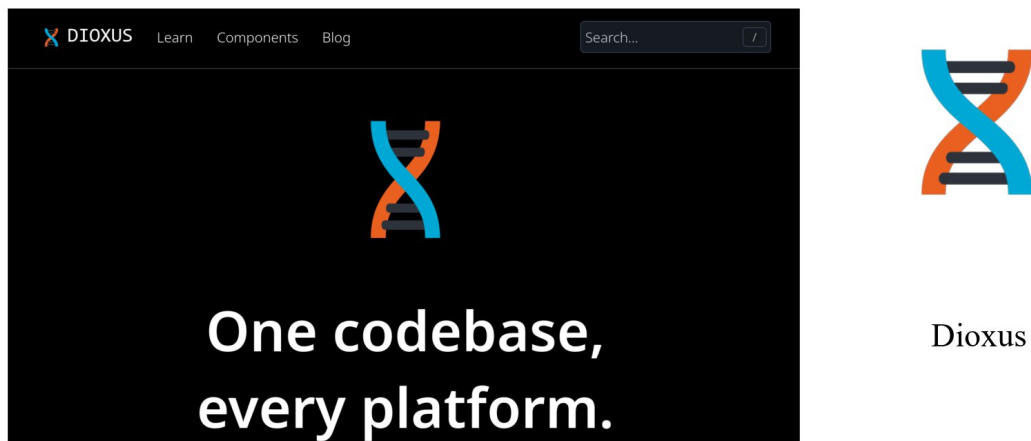


Рис 2.1. Головна сторінка веб сайту фреймворка “Tauri” за адресою “<https://v2.tauri.app/>”.

ГРАФІЧНІ БІБЛІОТЕКИ ТА ФРЕЙМОВРКИ



Рис 2.3 Приклад функції для створення елемента з допомогою макроса “rsx”.
Із сторінки за адресою “<https://dioxuslabs.com/learn/0.7/tutorial/rsx>”.

ГРАФІЧНІ БІБЛІОТЕКИ ТА ФРЕЙМОВРКИ

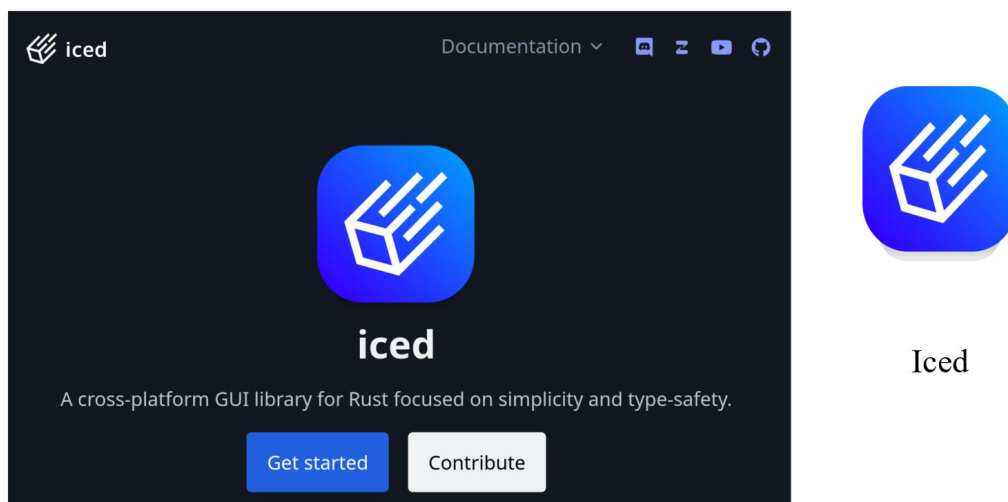


Рис 2.4 Головна сторінка бібліотеки “Iced” за адресою “https://iced.rs/”.

ГРАФІЧНІ БІБЛІОТЕКИ ТА ФРЕЙМОВРКИ

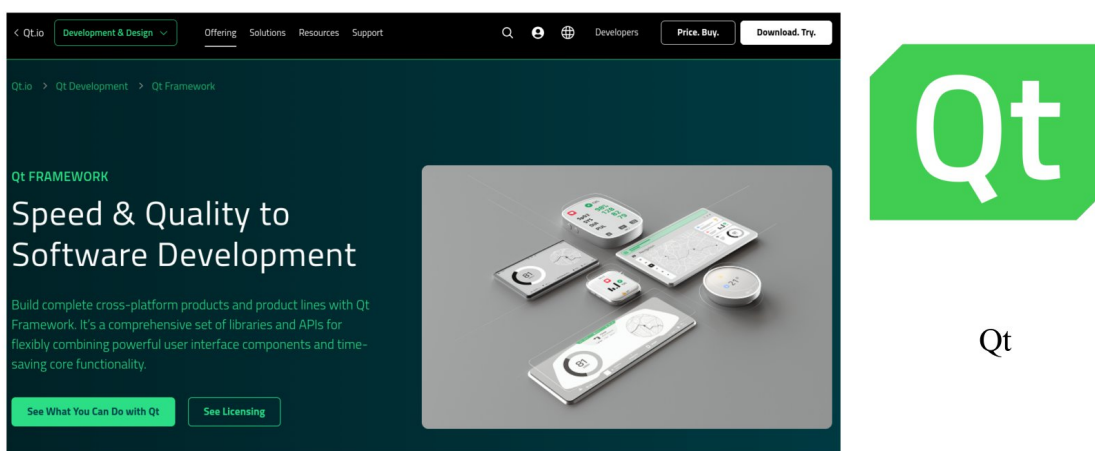
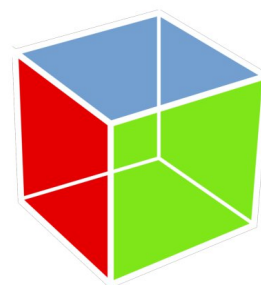
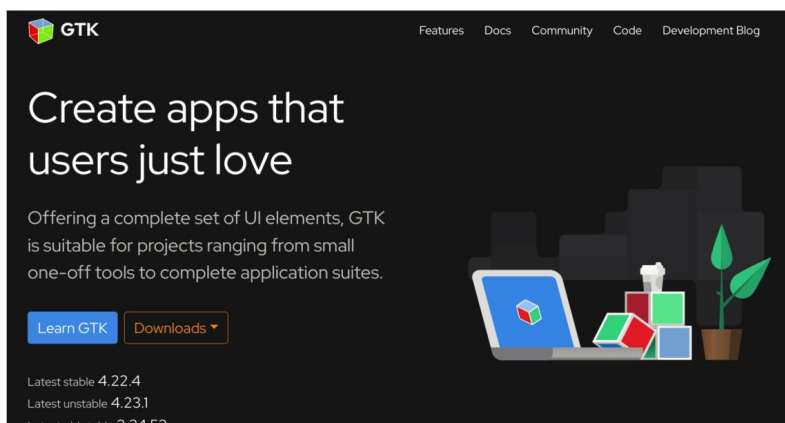


Рис 2.5 Головна сторінка фреймворка “Qt” за адресою “https://www.qt.io/development/qt-framework”.

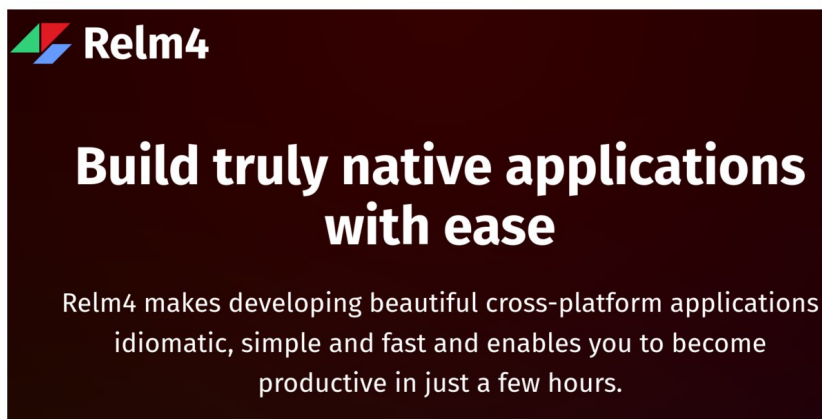
ГРАФІЧНІ БІБЛІОТЕКИ ТА ФРЕЙМОВРКИ



GTK

Рис 2.6 Головна сторінка проекту “GTK” за адресою “<https://www.gtk.org/>”.

ГРАФІЧНІ БІБЛІОТЕКИ ТА ФРЕЙМОВРКИ



Relm4

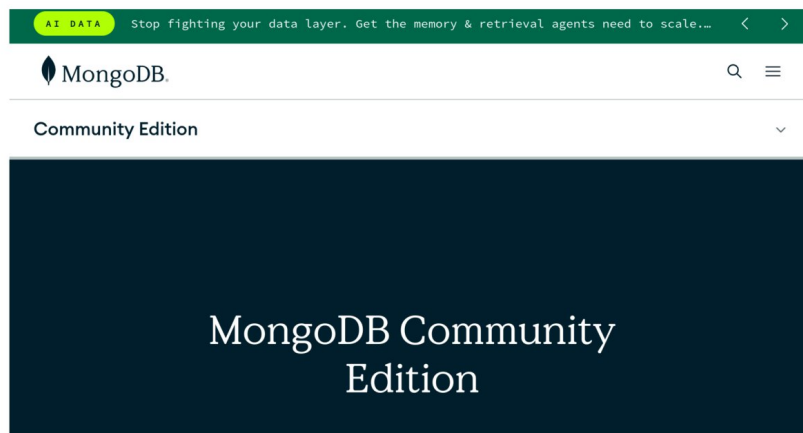
Рис 2.7 Головна сторінка бібліотеки “Relm4”, за адресою “<https://relm4.org/>”.

СИСТЕМИ БАЗ ДАНИХ



Рис 2.8 Головна сторінка бази даних “MySQL”, “<https://www.mysql.com/>”.

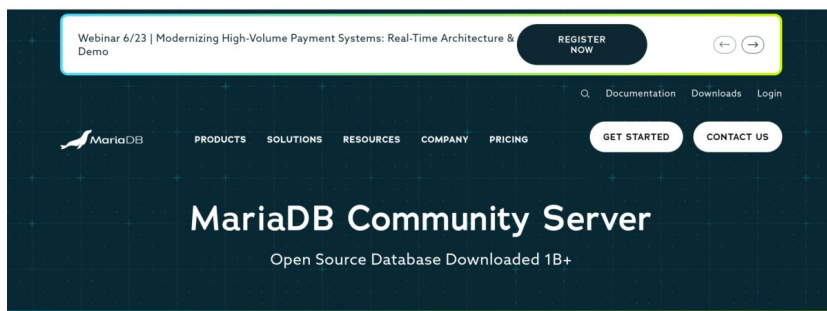
СИСТЕМИ БАЗ ДАНИХ



MongoDB

Рис 2.9 Головна сторінка бази даних MongoDB, “<https://www.mongodb.com/products/self-managed/community-edition>”.

СИСТЕМИ БАЗ ДАНИХ



MariaDB

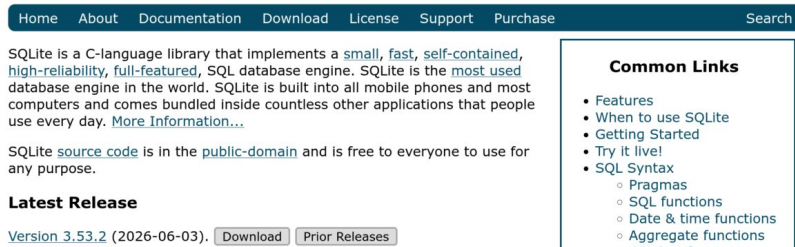
MariaDB Community Server is one of the most popular database servers in the world. The MariaDB database has been downloaded over 1 billion times and is the default over MySQL in the majority of Linux distributions. Created by MySQL's original developers, MariaDB is compatible with MySQL and Oracle, and is guaranteed to always be open source.



DOWNLOAD COMMUNITY SERVER

Рис 2.10 Головна сторінка бази даних “MariaDB”,
“https://mariadb.com/products/community-server/”.

СИСТЕМИ БАЗ ДАНИХ



SQLite

Рис 2.11 Головна сторінка бази даних “SQLite”.

ПОРІВНЯННЯ БАЗ ДАНИХ

Таблиця 2.1

База даних	Потреба розгортати локальний сервер	Має визначені типи колекцій
“MySQL”	Так	Так
“MariaDB”	Так	Так
“MongoDB”	Так	Ні
“SQLite”	Ні	Ні

АЛЬТЕРНАТИВА ЛОКАЛЬНІЙ ГЕНЕРАЦІЇ

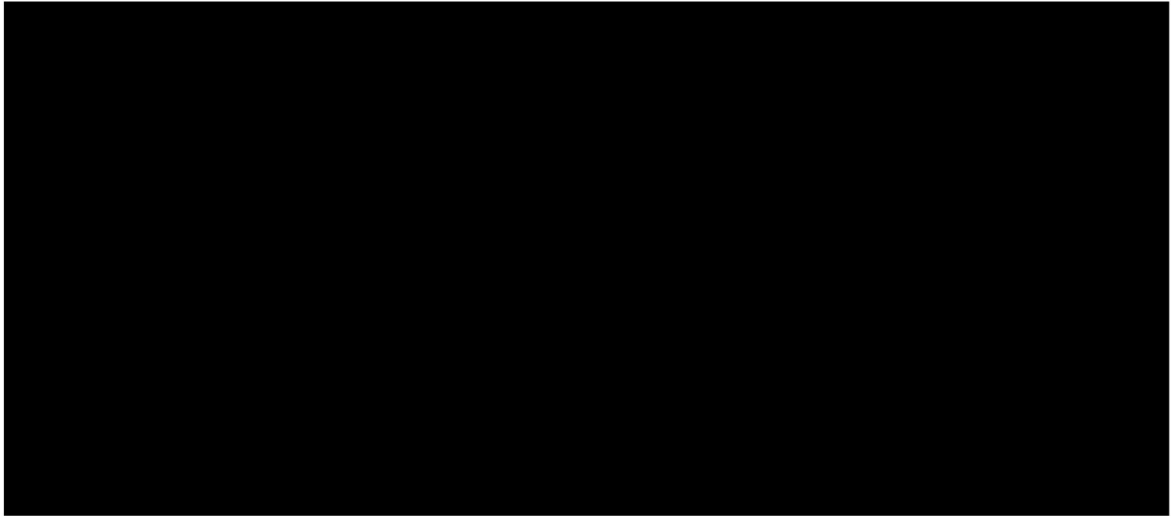


Бінарна серіалізація у Borsh



Текстова серіалізація у JSON

ВІДЕО-ДЕМОНСТРАЦІЯ РОБОТИ



КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
> src  
> target  
❖ .gitignore  
≡ Cargo.lock  
⚙ Cargo.toml  
⚙ rust-toolchain.toml
```

Рис 3.20 Вміст кореневого каталога проекту “mainuscript”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
[package]
name = "mainuscript"
version = "0.1.0"
edition = "2024"

[dependencies]
borsh = {version = "1.6.1", features = ["derive"]}
chrono = "0.4.44"
colored = "3.1.1"
figment = { version = "0.10.19", features = ["toml"]}
async-openai = { version = "0.40.3", features = [
    "_api",
    "response-types",
    "responses",
    "chat-completion-types",
    "chat-completion",
    "model",
    "model-types",
    "byot"
]}
regex = "1.12.3"
relm4 = "0.10.0"
serde = { version = "1.0.228", features = ["derive"] }
tracing = "0.1.44"
url = "2.5.8"
futures = "0.3.32"
```

Рис 3.21 Вміст файла "Cargo.toml".

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
fn main() {
    set_allowed_backends("x11,*");
    RELM_THREADS.set(8).unwrap();
    let app: RelmApp<TopInput> = RelmApp::new(app_id: "none.none.mainuscript");
    let input_config: InputConfig = match InputConfig::new(){
        Ok(input_config) => {input_config},
        Err(err_message: String) => {
            println!("{}", err_message);
            exit(code: -1)
        }
    };
    let bus: Bus = match Bus::new(input_config){
        Ok(bus: Bus) => {bus},
        Err(err_message: String) => {
            println!("{}", err_message);
            exit(code: -1)
        }
    };
    let bus_shared: Arc<RwLock<Bus>> = Arc::new(data: RwLock::new(bus));
    app.run:::TopView(payload: bus_shared.clone());
    match Arc::into_inner(this: bus_shared){
        Some(bus_rwlock: RwLock<Bus>) => {
            match bus_rwlock.into_inner(){
                Ok(bus: Bus) => {
                    bus.close();
                },
                Err(err: PoisonError<Bus>) => {
                    println!("{}", err.to_string());
                }
            }
        },
        None => {
            println!("{}", "Unexpected error happened when attempting to get bus shared reference to wrap things up");
        }
    }
}
} fn main
```

Рис 3.22 Вміст файла "main.rs".

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub struct Bus{
    pub css: Css,
    pub storage: Storage,
    pub config: Config,
    pub async_runtime: Runtime,
    pub ai: AI,
    pub options_data: Arc<RwLock<OptionsData>>
}
```

Рис 3.23 Структура “Bus” у файлі “bus.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub fn new(input_config: InputConfig) -> Result<Self, String>{
    let css: Css = Css::new();
    let storage: Storage = Storage::new(input_config.storage_folder);
    let serializable_config: SerializableConfig = match storage.read_from_file(file_path_input: CONFIG_FILE){ ...
    let config: Config = serializable_config.into_config();
    let async_runtime: Runtime = match Builder::new_multi_thread().enable_all().build(){ ...
    let ai: AI = AI::new(&config);
    let options_data_raw: OptionsData = match storage.read_from_file:<OptionsData>(file_path_input: SETTINGS_DATA_FILE) { ...
    let options_data: Arc<RwLock<OptionsData>> = Arc::new(data: RwLock::new(options_data_raw));
    Ok(Bus{
        css,
        storage,
        config,
        async_runtime,
        ai,
        options_data
    })
} fn new
```

Рис 3.24 Метод “new” структури “Bus” у файлі “bus.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub fn close(self){
    let storage_clone: Storage = self.storage.clone();
    let config_owned: Config = self.config;
    // storage_clone.store_to_file::<SerializableConfig>(&config_owned.into_serializable_config(), CONFIG_FILE);
    self.async_runtime.spawn_blocking(func: move ||{
        println!("First write entered");
        storage_clone.store_to_file::<SerializableConfig>[data: &config_owned.into_serializable_config(), file_path_input: CONFIG_
        println!("Second write exited");
    });
    let storage_owned: Storage = self.storage;
    let options_data_owned: Arc<RwLock<OptionsData>> = self.options_data;
    // storage_owned.store_to_file(&*options_data_owned.read().unwrap(), SETTINGS_DATA_FILE);
    self.async_runtime.spawn_blocking(func: move ||{
        println!("Second write entered");
        storage_owned.store_to_file(data: &*options_data_owned.read().unwrap(), file_path_input: SETTINGS_DATA_FILE);
        println!("Second write exited");
    });
    let runtime: Runtime = self.async_runtime;
    runtime.shutdown_timeout(Duration::from_hours(2));
}
```

Рис 3.25 Метод “close” структури “Bus” у файлі “bus.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
# colors.css
# logo.css
# navigation.css
# regular_chat.css
# settings.css
# top.css
```

Рис 3.26 Файли зі стилями CSS у каталозі “css”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
const CSS_TOP: &'static str = include_str!("./css/top.css");
const CSS_LOGO: &'static str = include_str!("./css/logo.css");
const CSS_NAVIGATION: &'static str = include_str!("./css/navigation.css");
const CSS_SETTINGS: &'static str = include_str!("./css/settings.css");
const CSS_REGULAR_CHAT: &'static str = include_str!("./css/regular_chat.css");
```

Рис 3.27 Стили CSS вбудовані у константи у файлі “css.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub fn new() -> Self {
    let css: String = String::from(CSS_TOP) +
        CSS_LOGO +
        CSS_NAVIGATION +
        CSS_SETTINGS +
        CSS_REGULAR_CHAT;

    let css_provider: CssProvider = CssProvider::new();

    css_provider.load_from_data(&css);
    style_context_add_provider_for_display(
        display: &Display::default().unwrap(),
        &css_provider,
        priority: gtk::STYLE_PROVIDER_PRIORITY_APPLICATION
    );
    Css { css_provider }
}
```

Рис 3.28 Метод “new” структури “Css” у файлі “css.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

Назва	Параметри	Призначення
new	На вхід: "PathBuf" з адресою домашнього каталога.	Створює новий екземпляр структури "Storage".
Назва	Параметри	Призначення
get_object_list (рис 3.29)	На вихід: колекція "Vec<FsObject>" або повідомлення "String" із помилкою.	Пробує прочитати список збережених чатів у домашньому каталозі.
create_and_populate_storage_dir (рис 3.30)	На вихід: повідомлення "String" із помилкою.	Пробує створити домашній каталог якщо він не існує.
read_from_file (рис 3.31)	На вхід: назва файла "&str"; на вихід: відповідний тип, або помилка "io::Error".	Пробує прочитати та десеріалізувати дані відповідного типу.
store_to_file (рис 3.32)	На вхід: дані що мають бути записані та назва файла "&str".	Пробує серіалізувати та записати дані у файл.
new_chat (рис 3.33)	На вхід: назва нового чата "&str"; на вихід: дані про файл нового чата або "io::Error".	Пробує створити нових файл для чата.

Таблиця 3.1
Методи сервіса "Storage".

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub fn get_object_list(&self) -> Result<Vec<FsObject>, String>{
    self.create_and_populate_storage_dir();
    let mut out_vec: Vec<FsObject> = Vec::with_capacity(16);
    let object_folder_iterator: ReadDir = match read_dir(path: &self.storage_folder.join(OBJECT_DIR_PATH))
    {
        Ok(iterator: ReadDir) => {iterator},
        Err(err: Error) => {
            return Err(format!("Error while opening object storage folder:\n{}", err.to_string()))
        }
    };
    for entry: Result<DirEntry, Error> in object_folder_iterator{
        // Ignore Errors
        if let Ok(dir_entry: DirEntry) = entry{
            // Ignore those errors too
            if let Ok(fs_object: FsObject) = FsObject::try_from(dir_entry){
                out_vec.push(fs_object);
            }
        }
    }
    return Ok(out_vec);
}
```

Рис 3.29 Метод get_object_list сервіса "Storage" у файлі "storage.rs".

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub fn create_and_populate_storage_dir(&self) -> OnFailure<String>{
    let objects_dir_path: PathBuf = PathBuf::from(&self.storage_folder).join(OBJECT_DIR_PATH);

    if let Err(err: Error) = create_dir_all(path: &self.storage_folder){
        #[cfg(test)] println!("{:?}", &err);
        return Some(err.to_string());
    }
    if let Err(err: Error) = create_dir_all(objects_dir_path){
        #[cfg(test)] println!("{:?}", &err);
        return Some(err.to_string());
    }
    return None;
}
```

Рис 3.30 метод `create_and_populate_storage_dir` сервіса “Storage” у файлі “storage.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub fn read_from_file<T: BorshDeserialize>(&self, file_path_input: &str) -> Result<T, std::io::Error>{
    let file_path: &PathBuf = &self.storage_folder.join(path: file_path_input);

    let file_result: Result<File, Error> = File::open(file_path);
    let output: T = match file_result{
        // If can open file
        Ok(mut file: File) => {
            match borsh::from_reader(&mut file) {
                Ok(config: T) => config,
                Err(err: Error) => {
                    return Err(err)
                }
            },
        // If failed to open a file
        Err(err: Error) => {return Err(err)}
    };
    return Ok(output)
}
```

Рис 3.31 метод `read_from_file` сервіса “Storage” у файлі “storage.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
fn store_to_file<T: BorshSerialize>(&self, data: &T, file_path_input: &str) -> OnFailure<std::io::Error>{
    let file_path: &PathBuf = &self.storage_folder.join(path: file_path_input);
    let file: File = match File::create(file_path){
        Ok(file: File) => {file},
        Err(err: Error) => {
            return Some(err);
        }
    };
    match borsh::to_writer(writer: file, value: &data){
        Ok(_) => return None,
        Err(err: Error) => {
            return Some(err);
        }
    }
}
```

Рис 3.32 метод store_to_file сервіса “Storage” у файлі “storage.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub fn new_chat(&self, name: NameString) -> Result<(RegularChat, FsObject), std::io::Error>{
    let chat_path: PathBuf = self.storage_folder.join(OBJECT_DIR_PATH).join::<String>(&
        path: format!("{}", name.inner).into()
    );
    let new_chat: RegularChat = RegularChat::new(name.clone());
    // IO part
    let file_path: &PathBuf = &self.storage_folder.join(chat_path.clone());
    let file: File = match File::create(file_path){
        Ok(file: File) => {file},
        Err(err: Error) => {
            return Err(err);
        }
    };
    if let Err(err: Error) = borsh::to_writer(writer: &file, value: &new_chat){
        return Err(err);
    }
    match self.store_to_file(data: &new_chat, file_path_input: &chat_path.to_string_lossy()){
        // ...
    }
} fn new_chat
```

Рис 3.33 метод new_chat сервіса “Storage” у файлі “storage.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub struct Config{  
    pub ai_api_url_key_pair: Arc<ConfigField<AiUrlKeyPair>>,  
    pub selected_model_name: Arc<ConfigField<Option<String>>>  
}
```

Рис 3.34 Структура “Config” у файлі “config.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub struct ConfigField<T: Sized + Send + Sync + Debug + Clone>{  
    // #[borsh(skip)]  
    inner: RwLock<T>,  
    subscribers: RwLock<HashMap<usize, Box<dyn Fn(&T) + Send + Sync>>>,  
}
```

Рис 3.35 Структура “ConfigField” у файлі “config_field.rs”

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub fn update(&self, value: T) -> T{
    let mut lock: RwLockWriteGuard<'_, T> = self.inner.write().expect("update in ConfigField");

    let old_value: T = std::mem::replace(dest: &mut *lock, src: value.clone());
    drop(lock);
    for (_, callback: &Box<dyn Fn(&T) + Send + Sync>) in self.subscribers.read().unwrap().iter(){
        callback(&value)
    };
    return old_value;
}
```

Рис 3.36 функція “update” структури “ConfigField” у файлі “config_field.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub type GlobalClientConfig = Arc<ConfigField<Option<Client<OpenAIConfig>>>>>;

#[derive(Debug)]
3 implementations
pub struct AI{
    openai: GlobalClientConfig,
    _ai_url_key_pair_subscriber: ConfigSubscription<AiUrlKeyPair>
}
```

Рис 3.37 Структура сервіса “AI” у файлі “ai.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```

let global_config_clone: Arc<ConfigField<Option<Client<...>>>> = global_config.clone();
let ai_url_key_pair_subscriber: ConfigSubscription<AiUrlKeyPair> = ConfigSubscription::subscribe(
    config.ai_api_url_key_pair.clone(), callback: Box::new(move |new_pair: &AiUrlKeyPair|{
        if let (Some(url: &Url), Some(key: &String)) = (&new_pair.url, &new_pair.key){
            let mut config: OpenAIConfig = OpenAIConfig::new();
            config = config.with_api_base(url.to_string());
            config = config.with_api_key(key);
            let client: Client<OpenAIConfig> = Client::with_config(config);
            global_config_clone.update(Some(client));
        }
        else {
            global_config_clone.update(None);
        }
    })
);

```

Рис 3.38 Фрагмент метода “new” сервіса “AI” який імплементує реакцію сервіса на зміну налаштування “ai_api_url_key_pair”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```

pub struct TopView{
    bus: Arc<RwLock<Bus>>,
    top_container: GtkBox,
    working_area: GtkBox,
    navigation_controller: Controller<NavigationView>,
    connector: Connectors
}

```

Рис. 3.39 Структура з даними компонента “TopVirew”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
let navigation_bar_controller: Controller<NavigationView> = navigation_bar_connector.forward(
    sender.input_sender(), transform: |navigation_output: NavigationOutput| {
        return TopInput::Navigation(navigation_output)
    });
```

Рис 3.40 Механізм що пересилає вихідне повідомлення від “NavigationView” до обробки компонентом “TopView” у файлі “top.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
NavigationView::Settings => {
    // Remove previous working area
    let previous_option: Option<Widget> = self.working_area.first_child();
    if let Some(previous: Widget) = previous_option {
        self.working_area.remove(child: &previous);
    }
    // Construct settings element
    let setting_builder: ComponentBuilder<SettingsView> = SettingsView::builder();
    let settings_root: &Box = &setting_builder.root;

    // Add settings view to the container
    self.working_area.append(child: settings_root);
    let setting_connector: Connector<SettingsView> = setting_builder.launch(payload: SettingsInit {
        bus: self.bus.clone(),
        highlighted_options: Arc::new(data: RwLock::new(0)),
        init_control_message: "".into()
    });
    self.connector = Connectors::Settings(setting_connector)
},
```

Рис 3.41 Алгоритм реакції компонента “TopView” на отримання повідомлення “NavigationView::Settings”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```

NavigationOutput::Object(fs_object: Arc<FsObject>) => {
    let regular_chat_builder: ComponentBuilder<RegularChatView> = RegularChatView::builder();
    let regular_chat_root: &Box = &regular_chat_builder.root;

    // Remove previous working area
    let previous_option: Option<Widget> = self.working_area.first_child();
    if let Some(previous: Widget) = previous_option {
        self.working_area.remove(child: &previous);
    }

    self.working_area.append(child: regular_chat_root);
    self.connector = Connectors::Chat(regular_chat_builder.launch(payload: RegularChatInit{
        bus: self.bus.clone(),
        chat_fs_object: fs_object
    }));
}

```

Рис 3.42 Обробка повідомлення “Object” компонентом “TopView”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```

pub struct SettingsView{
    bus: Arc<RwLock<Bus>>,
    widgets: SettingsWidgets,
    changed_options: Arc<RwLock<Selection>>, // All options whose value have changed
    error_options: Arc<RwLock<Selection>>, // All options that contain error
    highlighted_options: Arc<RwLock<Selection>> // All options that are highlighted
}

```

Рис 3.43 Структура “SettingsView”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```

{
    let sender_clone: ComponentSender<SettingsView> = sender.clone();
    save_button.connect_clicked(move |_| {
        sender_clone.input(message: SettingsInput::SaveAttempt);
    });
}
{
    let sender_clone: ComponentSender<SettingsView> = sender.clone();
    clear_button.connect_clicked(move |_| {
        sender_clone.input(message: SettingsInput::ClearAllOptions);
    });
}

```

Рис 3.44 Кнопки “Save” та “Clear” налаштовуються надсилати повідомлення про відповідну подію на обробку компоненту “SettingsView”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```

fn watch_entry_option(entry: &Entry, state_label: Label, sender: ComponentSender<SettingsView>, settings_data: &OptionsData) {
    entry.connect_changed(move |entry: &Entry| {
        let settings_data_ref: RwLockReadGuard<'_, OptionsData> = settings_data.read().unwrap();
        let Some(previous_text_wrapped: &OptionsDataKind) = settings_data_ref.get(&flag) else {return;};
        if let OptionsDataKind::String(previous_text: &String) = previous_text_wrapped {
            if *previous_text == entry.text().to_string() {
                set_state_label_empty(&state_label);
                sender.input(message: SettingsInput::SetOptionAsNotChanged(flag));
            }
            else {
                set_state_label_changed(&state_label);
                sender.input(message: SettingsInput::SetOptionAsChanged(flag));
            }
        }
    });
}

```

Рис 3.45 Функція watch_entry_option надсилає повідомлення про те змінено текстове поле чи ні.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub fn attempt_read_and_save_settings(settings_view: &mut SettingsView, sender: ComponentSender<SettingsView>){
    // Read raw values from widgets
    let api_url_raw: String = settings_view.widgets.option_widgets.api_url_entry.text().to_string();
    let api_key_raw: String = settings_view.widgets.option_widgets.api_key_entry.text().to_string();
    // Validate and convert values
    let api_url: Url = match validate_api_url(text: api_url_raw){
        Ok(data: Url) => data,
        Err(err: String) => {
            handle_invalid(settings_view, error: err);
            return;
        },
    };
    let api_key: String = match validate_api_key(text: api_key_raw){
        Ok(data: String) => data,
        Err(err: String) => {
            handle_invalid(settings_view, error: err);
            return;
        },
    };
    settings_view.widgets.control_label.set_text(str: "");
    // Save values to config
    let mut mut_bus: RwLockWriteGuard<'_, Bus> = settings_view.bus.write().unwrap();
    mut_bus.config.ai_api_url_key_pair.update(AiUrlKeyPair{
        url: Some(api_url),
        key: Some(api_key)
    });
    drop(mut_bus);
    // Save options data
    save_options_data(settings_view.bus.clone(), &settings_view.widgets);
    // Reload widgets
    sender.input(message: SettingsInput::ClearAllOptions);
}
fn attempt_read_and_save_settings
```

Рис 3.46 Функція attempt_read_and_save_settings яка запускається під час отримання повідомлення “SettingsInput::SaveAttempt” у файлі “ui/work_area/settings/save_implement.rs”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
SettingsInput::ClearAllOptions => {
    // self.widgets.clear_button.set_sensitive(false);
    self.widgets.root.remove(child: &self.widgets.scroll);
    let new_scroll: ScrolledWindow = scroll();
    let bus_ref: RwLockReadGuard<'_, Bus> = self.bus.read().unwrap();
    let new_option_widgets: OptionWidgets = create_entry_widgets(
        new_scroll.clone(),
        bus_ref.options_data.clone(),
        sender,
        self.highlighted_options.clone()
    );
    self.widgets.option_widgets = new_option_widgets;
    self.widgets.root.append(child: &new_scroll);
    self.widgets.scroll = new_scroll;
    self.widgets.clear_button.set_sensitive(false);
    self.widgets.save_button.set_sensitive(false);
}
```

Рис 3.47 Компонент “SettingsView” обробляє повідомлення “SettingsInput::ClearAllOptions”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
SettingsInput::SetOptionAsChanged(flag: u32) => {
    self.widgets.clear_button.set_sensitive(true);
    self.widgets.save_button.set_sensitive(true);
    self.changed_options.write().unwrap().set(flag);
},
```

Рис 3.48 Компонент “SettingsView” обробляє повідомлення
“SettingsInput::SetOptionAsChanged”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
SettingsInput::SetOptionAsNotChanged(flag: u32) => {
    let mut changed_options_ref: RwLockWriteGuard<'_, u32> = self.changed_options.write().unwrap();
    changed_options_ref.clear(flag);
    // If no changed settings remain
    if *changed_options_ref == 0 {
        self.widgets.clear_button.set_sensitive(false);
        self.widgets.save_button.set_sensitive(false);
    }
},
```

Рис 3.49 Компонент “SettingsView” обробляє повідомлення
“SettingsInput::SetOptionAsNotChanged”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
pub struct RegularChatViewData{
    status_label: GtkLabel,
    messages_scroll: ScrolledWindow,
    chat_entry: Entry,
    control_button: GtkButton,
    control_button_handler: SignalHandlerId,
    bus: Arc<RwLock<Bus>>,
    // ai_client_subscription: ConfigSubscription<Option<Client<OpenAIConfig>>>,
    selected_model_subscription: ConfigSubscription<Option<String>>,
    selected_model: String,
    ai_client: Option<Client<OpenAIConfig>>,
    chat_completion_request: CreateChatCompletionRequest,
    /// If this is set to false listening to AI response will stop
    message_is_read_from_stream: bool,
    /// Top Option is None when selected model did not change
    /// Top Option is Some when model was changed
    /// Internal Option is None when model was unselected
    /// Internal Option is Some when model was set to new one
    model_was_updated: Option<Option<String>>,
    messages_list: TypedListView<Message, NoSelection>,
    chat_data: RegularChat,
    chat_fs_object: Arc<FsObject>
}

impl RegularChatView{
pub struct RegularChatView{
    data: Option<RegularChatViewData>,
    is_uninit: bool
}
```

Рис 3.50 Компонент “RegularChatView”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
let control_button_handler: SignalHandlerId = control_button.connect_clicked(move |_|{
    sender_clone.input(message: RegularChatInput::SendMessage);
});
```

Рис 3.51 Кнопка надсилання повідомлення в чаті налаштовується надсилати повідомлення про подію “RegularChatInput::SendMessage”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```

let client_clone: Client<OpenAIConfig> = data.ai_client.clone().unwrap();
let chat_completion_request_clone: CreateChatCompletionRequest = data.chat_completion_request.clone();
sender.oneshot_command(future: async move{
    let chat_api: Chat<'_, OpenAIConfig> = client_clone.chat();
    let response_attempt: Result<CreateChatCompletionResponse, ...> = chat_api.create{
        request: chat_completion_request_clone
    }.await;
    match response_attempt{
        Ok(response: CreateChatCompletionResponse) => return RegularChatCommandOutput::GotResponse(response),
        Err(err: OpenAIError) => return RegularChatCommandOutput::Failure(err)
    }
});

```

Рис 3.52 Запуск асинхронної функції яка робить запит до “OpenAI” API.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```

fn update_cmd(
    message: Self::CommandOutput,
    sender: ComponentSender<Self>,
    root: &Self::Root,
) {
    match message{
        RegularChatCommandOutput::GotResponse(response: CreateChatCompletionResponse) => {
            if let Some(data: &mut RegularChatViewData) = &mut self.data{
                if data.message_is_read_from_stream{
                    for choice: ChatChoice in response.choices{
                        if let Some(content: String) = choice.message.content{
                            write_last_message_to_all_places(data, content);
                        }
                    }
                    sender.input(message: RegularChatInput::ResponseOver);
                }
                // When user hit stop button while waiting for response
            } else{
                delete_working_messages(data);
                data.status_label.set_label(str: "Stopped listening");
                set_controls_to_ready(data, sender);
            }
        },
    },
}

```

Рис 3.53 Частина функції “update_cmd” що обробляє успіх запиту.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```

RegularChatInput::ResponseOver => {
    if let Some(data: &mut RegularChatViewData) = &mut self.data{
        // Reset cutton and label
        set_controls_to_ready(data, sender.clone());

        // Set last update to now
        data.chat_data.last_update_date = chrono::Utc::now().timestamp_micros();

        // If model was unselected while previous response was listened in disable controls now
        if let Some(model_update: Option<String>) = data.model_was_updated.clone(){
            handle_unselected_model(new_model_status: model_update, data, sender);
            data.model_was_updated = None;
        }

        data.message_is_read_from_stream = false;
    }
}

```

Рис 3.54 Обробка повідомлення “RegularChatInput::ResponseOver”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```

RegularChatInput::StopMessage => {
    if let Some(data: &mut RegularChatViewData) = &mut self.data{
        data.message_is_read_from_stream = false;
    },
}

```

Рис 3.55 Обробка повідомлення “RegularChatInput::StopMessage”.

КОД ТА АРХІТЕКТУРА ПРОЕКТА

```
RegularChatInput::SelectedModelUpdated => {
    if let Some(data: &mut RegularChatViewData) = &mut self.data {
        let bus_ref: RwLockReadGuard<'_, Bus> = data.bus.read().unwrap();
        let new_model_status: Option<String> = bus_ref.config.selected_model_name.get_value_rwlock().read().unwrap().clone();
        drop(bus_ref);
        if data.message_is_read_from_stream {
            data.model_was_updated = Some(new_model_status);
        }
        else {
            handle_unselected_model(new_model_status, data, sender);
        }
    }
},
```

Рис 3.56 Обробка повідомлення “RegularChatInput::SelectedModelUpdated”.

АПРОБАЦІЯ

Дацко Миколай Анатолійович Імплементация бінарної серіалізації для власного типу за допомогою бібліотеки “Borsh” мови програмування “Rust” // Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. – К.: ДУІКТ, 2026.(подано до друку).

ВИСНОВКИ

У результаті виконання дипломної роботи було створено віконну програму за допомогою якої можна спілкуватися із текстовим штучним інтелектом із звичайного персонального комп'ютера.

У процесі виконання було проведено дослідження генерації текста за допомогою штучного інтелекта в межах якого було роз'яснено як хмарні провайдери штучного інтелекта організовані на високому рівні.

Було проведено пошук, та аналіз доступних машин висновування з відкритим кодом та ліцензією завдяки чому було здобуто розуміння того як штучний інтелект справді розгортається.

Було проведено пошук та аналіз аналогічних чат клієнтів в результаті чого здобуто розуміння різних форм у яких програми можуть підходити до генерації текста за допомогою штучного інтелекта, та окреслено різницю між різними способами поширення програм.

ВИСНОВКИ

Було проведено пошук та аналіз бібліотек та фреймворків для розробки графічного інтерфейса віконних програм для мови “Rust” в результаті чого було досягнуто складність вибору та різноманітність підходів.

Було проведено пошук та аналіз відомих баз даних в результаті чого було виведено мильні та слабкі сторони кожної з них та баз даних проти інших підходів взагалом.

Розроблено та застосовано методику поділу коду програми на частини використовуючи особливості мови програмування “Rust” та об'єктно-орієнтованого підхода.

Таким чином результатом виконання цієї роботи стала не лише одна програма а й купа навичок, знань та мудрості здобутої у процесі дослідження та розробки.

ДЯКУЮ ЗА УВАГУ!