ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

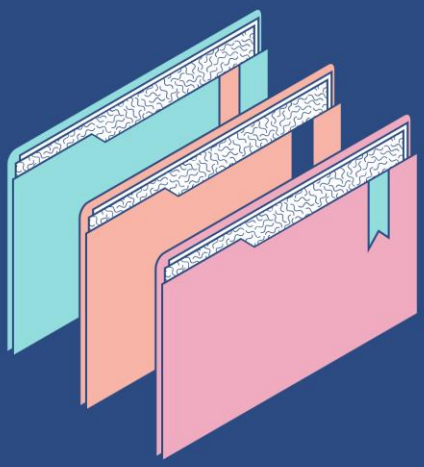
Веб-застосунок «AI Hub» для роботи зі штучним інтелектом

Виконав:

Дмитро Хабатілін
студ. гр. КІУКІ-21-6

Керівник:

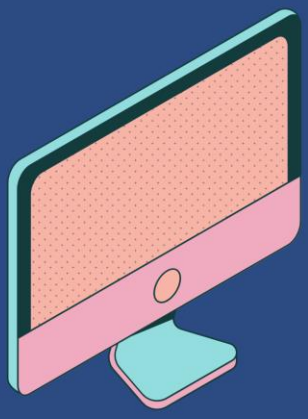
Наталія Бологова
доц. каф.ЕОМ



Мета та завдання

СТВОРЕННЯ ВЕБ-ЗАСТОСУНКУ «AI HUB», ЩО ПОЄДНУЄ МОВНІ МОДЕЛІ ТА МОДЕЛІ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ В ЄДИНОМУ ІНТЕРФЕЙСІ ДЛЯ ЗРУЧНОЇ ВЗАЄМОДІЇ З ІНСТРУМЕНТАМИ ШТУЧНОГО ІНТЕЛЕКТУ.

- Використання сучасного стеку технологій
- Розробка веб-застосунку для роботи з сучасними ШІ моделями
- Забезпечення можливості спілкування з мовними ШІ моделями
- Забезпечення генерації зображень ШІ моделями
- Забезпечення збереження чатів, повідомлень та облікових записів користувачів

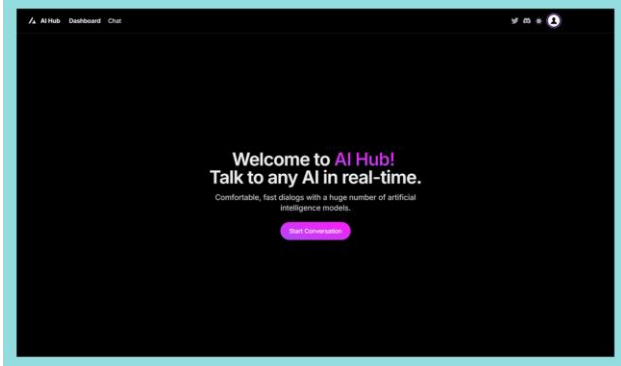


Актуальність

НЕЗРУЧНОСТІ ІСНУЮЧИХ ПЛАТФОРМ

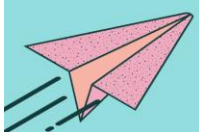
- Фрагментація інструментів ШІ
- Складнощі з обміном даними між платформами
- Витрати часу на вивчення різних сервісів
- Обмеженність платформ лише власними ШІ моделями

3



Переваги «AI Hub»

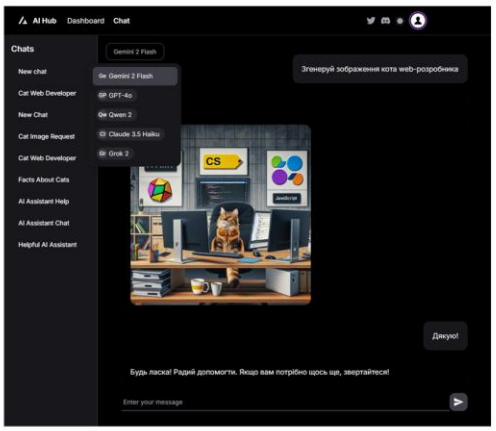
«AI Hub» вирішує ці проблеми, дозволяючи користувачам працювати з текстовими моделями та генерацією зображень в одному місці, без зайвих переходів і складних налаштувань.
Це економить час, спрощує взаємодію та робить ШІ доступним для всіх.



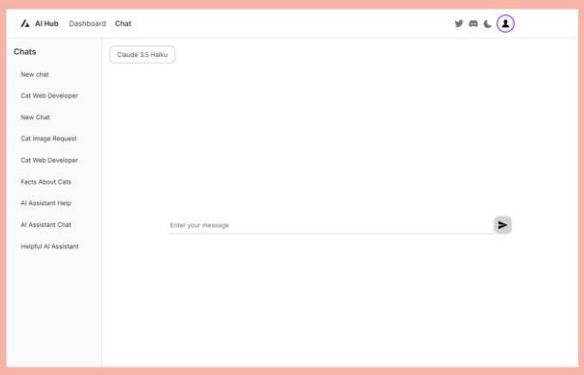
4

Основна функціональність

- Чат з LLM ШІ моделями у реальному часі
- Підтримка усіх популярних ШІ моделей
- Генерація зображень напряму у інтерфейсі чату
- Збереження історії, контексту



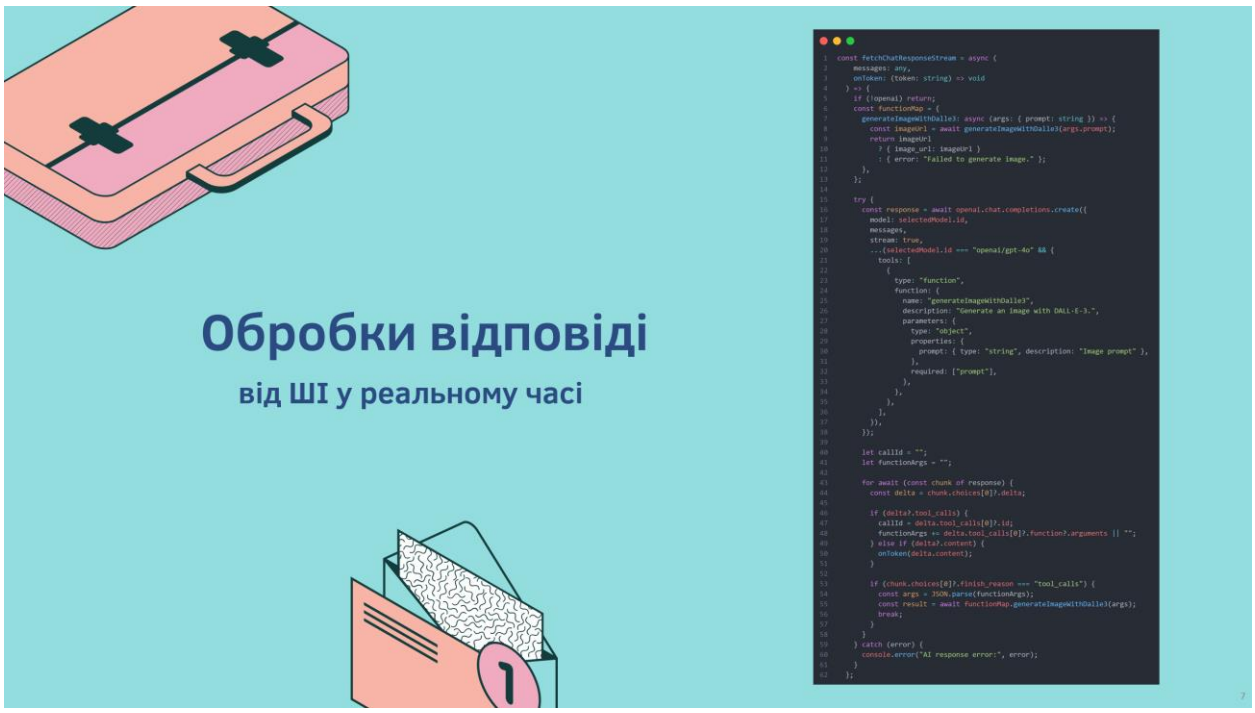
5



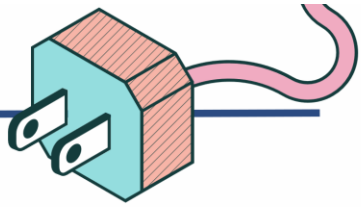
Чат – це серце «AI Hub»

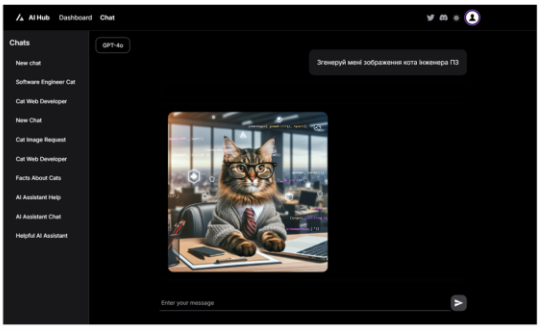
У верхній частині вікна користувач може обрати модель, наприклад, GPT-4 або Gemini, і одразу почати діалог.
Відповіді надходять у режимі реального часу завдяки стрімінгу, що створює відчуття живого спілкування. Поле для введення запитів просте та інтуїтивне, дозволяючи задавати як короткі питання, так і складні завдання. Такий підхід робить взаємодію з ШІ швидкою та природною, без зайвих налаштувань.

6



Генерація зображень



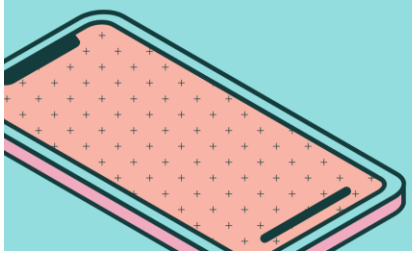


Однією з ключових особливостей «AI Hub» є генерація зображень прямо в чаті.

Користувач може написати, наприклад, «Згенеруй кота в стилі кіберпанк», і модель автоматично викликає генерацію зображення. Зображення з'являється в потоці діалогу, як звичайна відповідь, без потреби переходити до іншого сервісу.

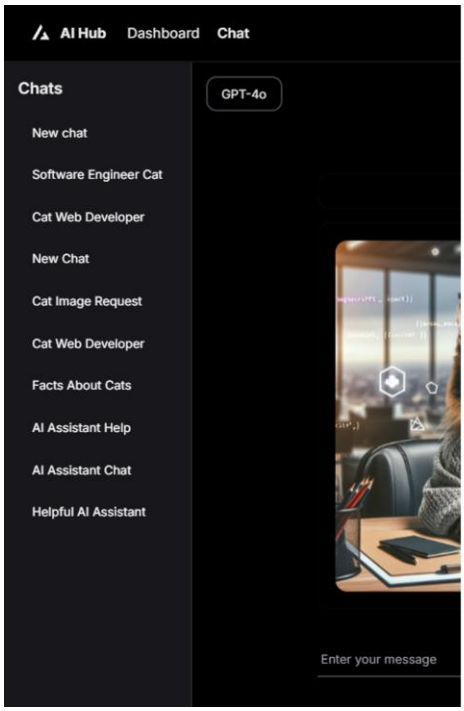
Це забезпечує безперервний мультимодальний досвід, де текст і зображення поєднуються в одному інтерфейсі.

Генерація зображень



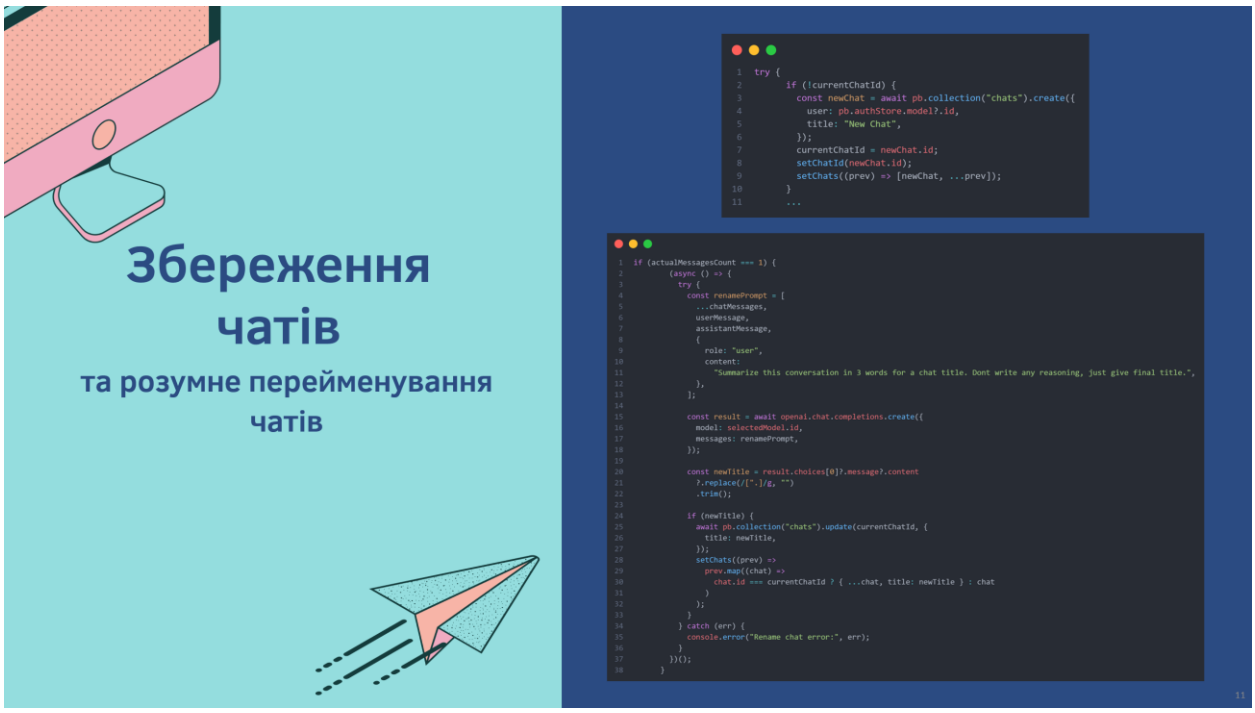
```
const generateImageWithDalle3 = async (prompt: string) => {
  try {
    setGeneratingImage(true);
    const response = await dalle3.images.generate({
      model: "dall-e-3",
      prompt,
      n: 1,
      size: "1024x1024",
    });
    const imageUrl = response.data[0]?.url;
    if (imageUrl) {
      const imgMsg = {
        role: "assistant",
        content: imageUrl,
        isImage: true,
      };
      setChatMessages((prev) => [...prev, imgMsg]);
      await saveMessage(imgMsg);
    }
    return imageUrl;
  } catch (e) {
    console.error("Image generation failed:", e);
    return null;
  } finally {
    setGeneratingImage(false);
  }
};
```

```
const saveMessage = async (msg: ChatMessage) => {
    if (!chatId) return;
    await pb.collection("messages").create({
      chat: chatId,
      role: msg.role,
      content: msg.content,
      model: selectedModel.id,
      isImage: !!msg.isImage,
    });
  };
```

Збереження історії чатів

«AI Hub» дозволяє зберігати всю історію взаємодії з ШІ. Кожен чат автоматично записується в базу даних PocketBase, прив'язуючись до профілю користувача.
Система сама генерує короткі назви для чатів на основі їхнього вмісту, що полегшує навігацію. Користувач може будь-коли повернутися до попередньої сесії, переглянути діалог або продовжити роботу. Це ідеально для аналізу відповідей чи повторного використання результатів, підвищуючи зручність і продуктивність.


Збереження чатів
та розумне перейменування чатів
try {
if (!currentChatId) {
const newChat = await pb.collection("chats").create({
user: pb.authStore.model?.id,
title: "New Chat",
});
currentChatId = newChat.id;
setChatId(newChat.id);
setChats((prev) => [newChat, ...prev]);
}
...
if (actualMessagesCount === 1) {
(async () => {
try {
const renamePrompt = [
...chatMessages,
userMessage,
assistantMessage,
{
role: "user",
content:
"Summarize this conversation in 3 words for a chat title. Dont write any reasoning, just give final title.",
},
];
const result = await openai.chat.completions.create({
model: selectedModel.id,
messages: renamePrompt,
});
const newTitle = result.choices[0]?.message?.content
?.replace(/["']/g, "")
.trim();
if (newTitle) {
await pb.collection("chats").update(currentChatId, {
title: newTitle,
});
setChats((prev) =>
prev.map((chat) =>
chat.id === currentChatId ? { ...chat, title: newTitle } : chat
)
);
}
} catch (err) {
console.error("Rename chat error:", err);
}
})();
}


Фронтенд-стек
Фронтенд «AI Hub» побудовано на Next.js із використанням React для створення швидкого та адаптивного інтерфейсу.
• Next.js і React
NEXT.JS
• NextUI для UI
• TypeScript
TS

Бекенд-стек

Бекенд «AI Hub» реалізовано на PocketBase – легкій і швидкій базі даних із вбудованим API.
PocketBase відповідає за:

- збереження історії чатів
- управління профілями користувачів
- аутентифікацію

Його простота дозволяє швидко розгортати сервер і масштабувати застосунок.

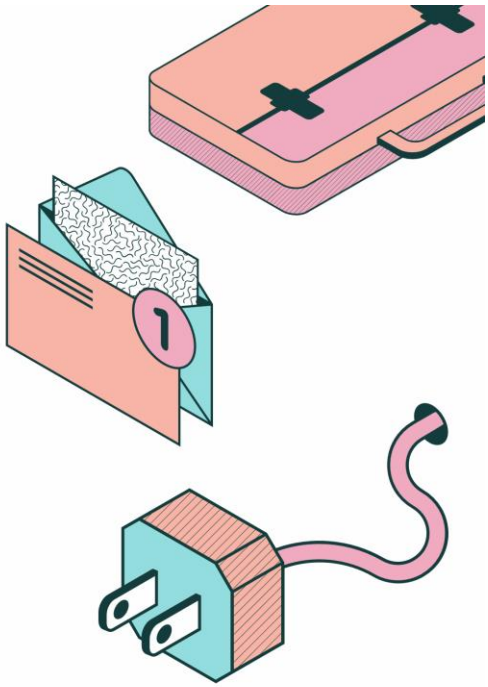
13

API для ШІ

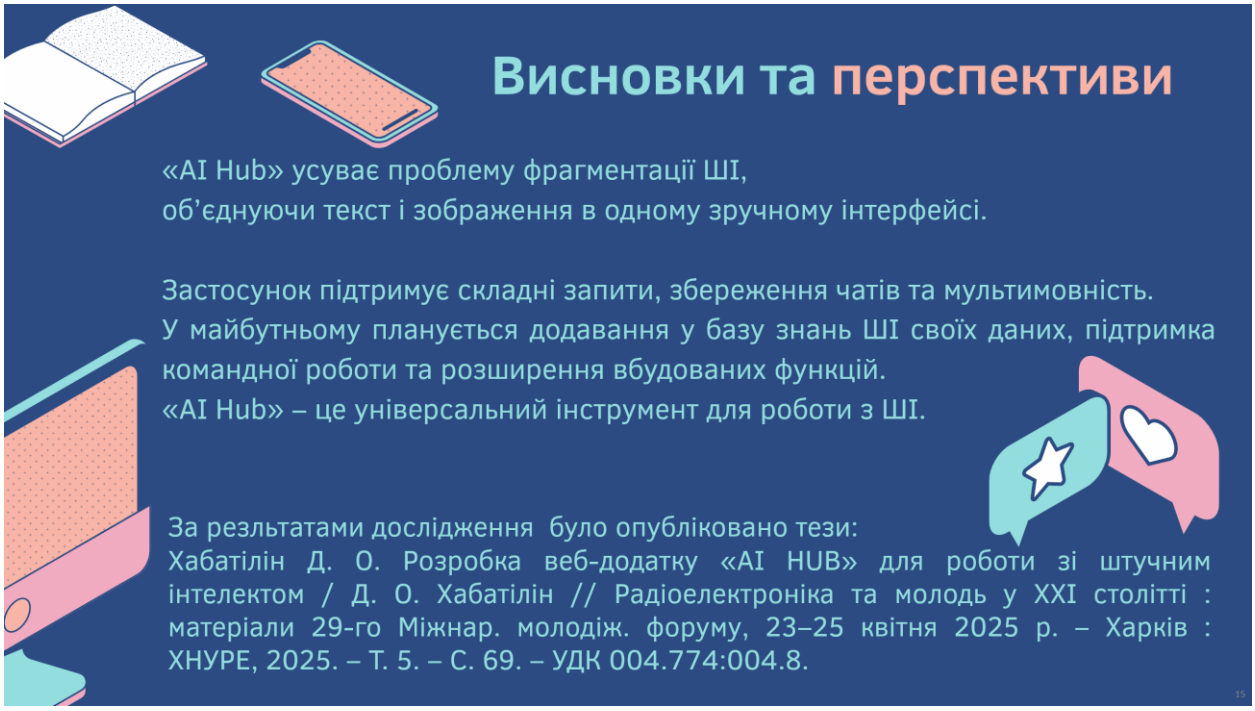
Для роботи з ШІ «AI Hub» використовує два ключові API:

- OpenRouter API
- OpenAI API

OpenRouter забезпечує доступ до текстових моделей, таких як GPT-4 і Claude тощо, із підтримкою стрімінгу для миттєвих відповідей. OpenAI API відповідає за генерацію зображень.



14


Висновки та перспективи
«AI Hub» усуває проблему фрагментації ШІ,
об'єднуючи текст і зображення в одному зручному інтерфейсі.
Застосунок підтримує складні запити, збереження чатів та мультимовність.
У майбутньому планується додавання у базу знань ШІ своїх даних, підтримка командної роботи та розширення вбудованих функцій.
«AI Hub» – це універсальний інструмент для роботи з ШІ.
За результатами дослідження було опубліковано тези:
Хабатілін Д. О. Розробка веб-додатку «AI HUB» для роботи зі штучним інтелектом / Д. О. Хабатілін // Радіоелектроніка та молодь у XXI столітті : матеріали 29-го Міжнар. молодіж. форуму, 23–25 квітня 2025 р. – Харків : ХНУРЕ, 2025. – Т. 5. – С. 69. – УДК 004.774:004.8.

ДОДАТОК Б

Вихідний код

Б.1 Сторінка Chat

```
"use client";
import { useEffect, useState } from "react";
import { Button } from "@nextui-org/button";
import { Textarea } from "@nextui-org/input";
import { Card, CardBody } from "@nextui-org/card";
import { Image } from "@nextui-org/image";
import { Chip } from "@nextui-org/chip";
import { Spinner } from "@nextui-org/spinner";
import { Avatar } from "@nextui-org/avatar";
import {
  Dropdown,
  DropdownTrigger,
  DropdownMenu,
  DropdownItem,
} from "@nextui-org/dropdown";
import {
  Modal,
  ModalContent,
  ModalHeader,
  ModalBody,
  useDisclosure,
} from "@nextui-org/modal";
import ReactMarkdown from "react-markdown";
import remarkGfm from "remark-gfm";
import { ChatMessage } from
"../interfaces/ChatMessage.interface";
import useAuthRedirect from "@/hooks/useAuthRedirect";
import pb from "@/lib/pocketbase";
import SendIcon from "@mui/icons-material/Send";
import Sidebar from "@/components/sidebar";
import { openai, dalle3, initializeOpenAI } from
"@/app/core/initOpenAI";
import { models } from "@/app/core/models";
let currentChatId: string | null = null;

export default function Chat() {
  useAuthRedirect();

  const [chats, setChats] = useState<any[]>([]);
  const [chatId, setChatId] = useState<string | null>(null);
  const [showSidebar, setShowSidebar] = useState(false);
```

```
  useEffect(() => {
    let isMounted = true;
    const fetchChats = async () => {
      try {
        const result = await
pb.collection("chats").getFullList({
          filter: `user="${pb.authStore.model?.id}"`,
          sort: "-created",
        });
        if (isMounted) setChats(result);
      } catch (error) {
        if (isMounted) console.error("Failed to load chats:",
error);
      }
    };
    fetchChats();
    return () => {
      isMounted = false;
    };
  }, []);

  useEffect(() => {
    const handleResize = () => {
      if (window.innerWidth < 768) {
        setShowSidebar(false);
      } else {
        setShowSidebar(true);
      }
    };
    handleResize();
    window.addEventListener("resize", handleResize);
    return () => window.removeEventListener("resize",
handleResize);
  }, []);

  const { isOpen, onOpen, onClose } = useDisclosure();
  const [currentImage, setCurrentImage] = useState<string |
null>(null);
  const [prompt, setPrompt] = useState("");
  const [chatMessages, setChatMessages] =
useState<ChatMessage[]>([
    { role: "system", content: "You are an AI assistant." },
  ]);
  const [loading, setLoading] = useState(false);
  const [generatingImage, setGeneratingImage] = useState(false);
  const [selectedModel, setSelectedModel] = useState(models[0]);

  useEffect(() => {
    initializeOpenAI();
  }, []);

  const handleSelectChat = async (id: string) => {
    setChatId(id);
```

```
    if (!id) {
      setChatMessages([{ role: "system", content: "You are an AI
assistant." }]);
      return;
    }

    try {
      const messages = await
pb.collection("messages").getFullList({
        filter: `chat="${id}" && content != ""`,
        sort: "created",
      });

      const mapped = messages.map((msg) => ({
        role: msg.role,
        content: msg.content,
        isImage: msg.isImage || false,
      }));

      setChatMessages([{ role: "system", content: "You are an AI
assistant." }, ...mapped]);
    } catch (err) {
      console.error("Failed to load chat messages:", err);
    }
  };

  const saveMessage = async (msg: ChatMessage, chatIdToUse?:
string) => {
    if (!chatId && !chatIdToUse) return;
    await pb.collection("messages").create({
      chat: chatId,
      role: msg.role,
      content: msg.content,
      model: selectedModel.id,
      isImage: !!msg.isImage,
    });
  };

  const handleSendMessage = async () => {
    if (!prompt.trim()) return;

    const userMessage = { role: "user", content: prompt };
    const newMessages = [...chatMessages, userMessage];
    setChatMessages(newMessages);
    setPrompt("");
    setLoading(true);

    currentChatId = chatId;

    try {
      if (!currentChatId) {
        const newChat = await pb.collection("chats").create({
          user: pb.authStore.model?.id,
```

```
          title: "New Chat",
        });

        currentChatId = newChat.id;
        setChatId(currentChatId);
        setChats((prev) => [newChat, ...prev]);
      }

      await pb.collection("messages").create({
        chat: currentChatId,
        content: prompt,
        role: "user",
        model: selectedModel.id,
      });

      let assistantMessage = { role: "assistant", content: "" };
      setChatMessages([...newMessages, assistantMessage]);

      await fetchChatResponseStream(newMessages, async (token)
=> {
        assistantMessage.content += token;
        setChatMessages([...newMessages, assistantMessage]);
      });

      await pb.collection("messages").create({
        chat: currentChatId,
        content: assistantMessage.content,
        role: "assistant",
        model: "google/gemini-2.0-flash-exp:free",
      });
      const actualMessagesCount = newMessages.filter(
        (m) => m.role !== "system"
      ).length;
      if (actualMessagesCount === 1) {
        (async () => {
          try {
            const renamePrompt = [
              ...chatMessages,
              userMessage,
              assistantMessage,
              {
                role: "user",
                content:
                  "Summarize this conversation in 3 words for a
chat title. Dont write any reasoning, just give final title.",
              },
            ];

            const result = await
openai.chat.completions.create({
              model: selectedModel.id,
              messages: renamePrompt,
            });
```

```
            const newTitle = result.choices[0]?.message?.content
              ?.replace(/["."]/g, "")
              .trim();

            if (newTitle) {
              await pb.collection("chats").update(currentChatId,
{
                title: newTitle,
              });
              setChats((prev) =>
                prev.map((chat) =>
                  chat.id === currentChatId ? { ...chat, title:
newTitle } : chat
                )
              );
            }
          } catch (err) {
            console.error("Rename chat error:", err);
          }
        })();
      }
    } catch (error) {
      console.error("Send error:", error);
    } finally {
      setLoading(false);
    }
  };

  const fetchChatResponseStream = async (
    messages: any,
    onToken: (token: string) => void
  ) => {
    if (!openai) return;
    const functionMap = {
      generateImageWithDalle3: async (args: { prompt: string })
=> {
        const imageUrl = await
generateImageWithDalle3(args.prompt);
        return imageUrl
          ? { image_url: imageUrl }
          : { error: "Failed to generate image." };
      },
    };

    try {
      const response = await openai.chat.completions.create({
        model: selectedModel.id,
        messages,
        stream: true,
        ...(selectedModel.id === "openai/gpt-4o" && {
          tools: [
            {
```

```
              type: "function",
              function: {
                name: "generateImageWithDalle3",
                description: "Generate an image with DALL·E-3.",
                parameters: {
                  type: "object",
                  properties: {
                    prompt: { type: "string", description:
"Image prompt" },
                  },
                  required: ["prompt"],
                },
              },
            },
          ],
        }),
      });

      let callId = "";
      let functionArgs = "";

      for await (const chunk of response) {
        const delta = chunk.choices[0]?.delta;

        if (delta?.tool_calls) {
          callId = delta.tool_calls[0]?.id;
          functionArgs +=
delta.tool_calls[0]?.function?.arguments || "";
        } else if (delta?.content) {
          onToken(delta.content);
        }

        if (chunk.choices[0]?.finish_reason === "tool_calls") {
          const args = JSON.parse(functionArgs);
          const result = await
functionMap.generateImageWithDalle3(args);
          break;
        }
      }
    } catch (error) {
      console.error("AI response error:", error);
    }
  };

  const generateImageWithDalle3 = async (prompt: string) => {
    try {
      setGeneratingImage(true);
      const response = await dalle3.images.generate({
        model: "dall-e-3",
        prompt,
        n: 1,
        size: "1024x1024",
      });
```

```
      const imageUrl = response.data[0]?.url;
      if (imageUrl) {
        const imgMsg = {
          role: "assistant",
          content: imageUrl,
          isImage: true,
        };
        setChatMessages((prev) => [...prev, imgMsg]);
        await saveMessage(imgMsg, currentChatId!);
      }
      return imageUrl;
    } catch (e) {
      console.error("Image generation failed:", e);
      return null;
    } finally {
      setGeneratingImage(false);
    }
  };

  const hasMessages = chatMessages.some((msg) => msg.role !==
"system");

  return (
    <div className="flex h-full w-full overflow-hidden pl-4 pr-
4">
      {showSidebar && (
        <Sidebar chats={chats} onSelectChat={(id) => {
          handleSelectChat(id);
          if (window.innerWidth < 768) setShowSidebar(false);
        }} />
      )}
      {window.innerWidth < 768 && (
        <Button
          variant="solid"
          color="default"
          isIconOnly
          className="absolute top-4 right-4 z-50"
          onClick={() => setShowSidebar(!showSidebar)}
        >
          ☰
        </Button>
      )}
      {!(window.innerWidth < 768 && showSidebar) && (
        <div className="relative flex flex-col justify-center
items-center w-full py-8 h-full gap-5">
          <div className="absolute top-4 left-4 z-50">
            <Dropdown>
              <DropdownTrigger>
                <Button
variant="solid">{selectedModel.name}</Button>
              </DropdownTrigger>
              <DropdownMenu
                aria-label="Select AI Model"
```

```
                selectionMode="single"
                onAction={(key) => {
                  const selected = models.find((model) =>
model.id === key);
                  if (selected) setSelectedModel(selected);
                }}
              >
                {models.map((model) => (
                  <DropdownItem key={model.id}>
                    <Chip
                      avatar={
                        <Avatar
                          getInitials={(name) => name.slice(0,
2)}
                          name={model.name}
                          size="sm"
                        />
                      }
                      variant="flat"
                    >
                      {model.name}
                    </Chip>
                  </DropdownItem>
                ))}
              </DropdownMenu>
            </Dropdown>
          </div>

          <div
            className={`flex flex-col gap-4 w-full max-w-3xl
overflow-y-auto ${hasMessages ? "flex-grow" : "justify-
center"}`}
            style={{ paddingBottom: "80px", maxHeight:
"calc(100vh - 150px)" }}
          >
            {chatMessages.map(
              (msg, index) =>
                msg.role !== "system" && (
                  <div
                    key={index}
                    className={`${msg.role === "user" ? "text-
right ml-auto" : "text-left"}`}
                  >
                    <Card
                      shadow="sm"
                      className={`p-2 ${msg.role === "assistant"
? "w-full" : "w-max"} ${msg.role === "assistant" && "bg-
transparent"}`}
                    >
                      <CardBody>
                        {msg.isImage ? (
                          <Image
                            alt="Generated Image"
```

```
                            src={msg.content}
                            shadow="lg"
                            isZoomed
                            width={400}
                            onClick={() => {
                              setCurrentImage(msg.content);
                              onOpen();
                            }}
                          />
                        ) : (
                          <ReactMarkdown
remarkPlugins={[remarkGfm]}>
                            {msg.content}
                          </ReactMarkdown>
                        )}
                      </CardBody>
                    </Card>
                    {generatingImage && index ===
chatMessages.length - 1 && (
                      <Chip
                        size="md"
                        variant="bordered"
                        className="flex pt-6 pb-6 pr-4 justify-
items-center items-center"
                      >
                        <Spinner size="sm" color="primary" />
                        <span className="ml-2">The image is
generating...</span>
                      </Chip>
                    )}
                  </div>
                )
            )}
          </div>

          <div
            className={`flex flex-row items-center w-full max-w-
3xl ${hasMessages ? "fixed bottom-4" : ""}`}
            style={{ paddingBottom: "10px" }}
          >
            <Textarea
              value={prompt}
              onValueChange={setPrompt}
              variant="flat"
              placeholder="Enter your message"
              minRows={1}
              className="flex-grow py-2 ml-4 mr-4"
              disabled={loading}
            />
            <Button onClick={handleSendMessage} isIconOnly
isDisabled={loading}>
              <SendIcon />
            </Button>
```

```
          </div>
        </div>
    )}


      <Modal
        isOpen={isOpen}
        onOpenChange={onClose}
        backdrop="blur"
        size="2xl"
        shouldBlockScroll
      >
        <ModalContent className="pb-3">
          <ModalHeader />
          <ModalBody>
            {currentImage && (
              <Image
                src={currentImage}
                radius="sm"
                alt="Large generated image"
                width="100%"
              />
            )}
          </ModalBody>
        </ModalContent>
      </Modal>
    </div>
  );
}
```

Б.2 Провайдери API

```
import OpenAI from "openai";

export let openai: any;
export let dalle3: any;

export async function initializeOpenAI() {
  openai = new OpenAI({
    dangerouslyAllowBrowser: true,
    baseURL: "https://openrouter.ai/api/v1",
    apiKey: process.env.NEXT_PUBLIC_OPENROUTER_API_KEY!,
    defaultHeaders: {
      "HTTP-Referer": "https://your-site-url.com",
      "X-Title": "AI Hub",
    },
  });

  dalle3 = new OpenAI({
    dangerouslyAllowBrowser: true,
    baseURL: "https://api.openai.com/v1",
    apiKey: process.env.NEXT_PUBLIC_DALLE_API_KEY!,
```

```
  });

  return { openai, dalle3 };
}
```

Б.3 Вебхук перевірки авторизації

```
"use client";

import { useEffect } from "react";
import { useRouter } from "next/navigation";
import pb from "@/lib/pocketbase";

export default function useAuthRedirect() {
  const router = useRouter();

  useEffect(() => {
    if (!pb.authStore.isValid) {
      router.replace("/auth");
    }
  }, []);
}
```

Б.4 Сторінка Auth

```
"use client";
import { useState } from "react";
import { Input } from "@nextui-org/input";
import { Button } from "@nextui-org/button";
import { Card, CardBody } from "@nextui-org/card";
import { Tabs, Tab } from "@nextui-org/tabs";
import { Logo } from "@/components/icons";
import { register, login } from "@/lib/auth";
import { useRouter } from "next/navigation";

export default function AuthPage() {
  const router = useRouter();
  const [loginForm, setLoginForm] = useState({ email: "",
password: "" });
  const [registerForm, setRegisterForm] = useState({ username:
"", email: "", password: "" });
  const [loading, setLoading] = useState(false);
  const handleLogin = async () => {
    try {
      setLoading(true);
      await login(loginForm.email, loginForm.password);
      console.log("Logged in");
      router.push("/chat");
    } catch (err) {
      console.error("Login failed:", err);
    } finally {
      setLoading(false);
```

```
    }
  };

  const handleRegister = async () => {
    try {
      setLoading(true);
      await register(
        registerForm.email,
        registerForm.username,
        registerForm.password,
    registerForm.password
      );
      console.log("Registered");
      await login(registerForm.email, registerForm.password);
      console.log("Logged in");
      router.push("/chat");
    } catch (err) {
      console.error("Registration failed:", err);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="flex justify-center items-center h-screen
bg-background px-4">
      <Card className="max-w-md w-full p-4">
        <CardBody>
          <div className="flex items-center gap-2">
            <Logo />
            <p className="font-bold text-inherit">AI Hub</p>
          </div>
          <Tabs fullWidth>
            <Tab key="login" title="Login">
              <form className="flex flex-col gap-4"
onSubmit={(e) => { e.preventDefault(); handleLogin(); }}>
                <Input
                  type="email"
                  label="Email"
                  value={loginForm.email}
                  onValueChange={(val) => setLoginForm({
...loginForm, email: val })}
                  required
                />
                <Input
                  type="password"
                  label="Password"
                  value={loginForm.password}
                  onValueChange={(val) => setLoginForm({
...loginForm, password: val })}
                  required
                />
                <Button
```

```
                  type="submit"
                  className="bg-gradient-to-tr from-
customViolet2 to-customViolet1 text-white shadow-lg"
                  fullWidth
                  isLoading={loading}
                >
                  Sign In
                </Button>
              </form>
            </Tab>
            <Tab key="register" title="Register">
              <form className="flex flex-col gap-4"
onSubmit={(e) => { e.preventDefault(); handleRegister(); }}>
                <Input
                  label="Username"
                  value={registerForm.username}
                  onValueChange={(val) => setRegisterForm({
...registerForm, username: val })}
                  required
                />
                <Input
                  type="email"
                  label="Email"
                  value={registerForm.email}
                  onValueChange={(val) => setRegisterForm({
...registerForm, email: val })}
                  required
                />
                <Input
                  type="password"
                  label="Password"
                  value={registerForm.password}
                  onValueChange={(val) => setRegisterForm({
...registerForm, password: val })}
                  required
                />
                <Button
                  type="submit"
                  className="bg-gradient-to-tr from-
customViolet2 to-customViolet1 text-white shadow-lg"
                  fullWidth
                  isLoading={loading}
                >
                  Sign Up
                </Button>
              </form>
            </Tab>
          </Tabs>
        </CardBody>
      </Card>
    </div>
  );
}
```