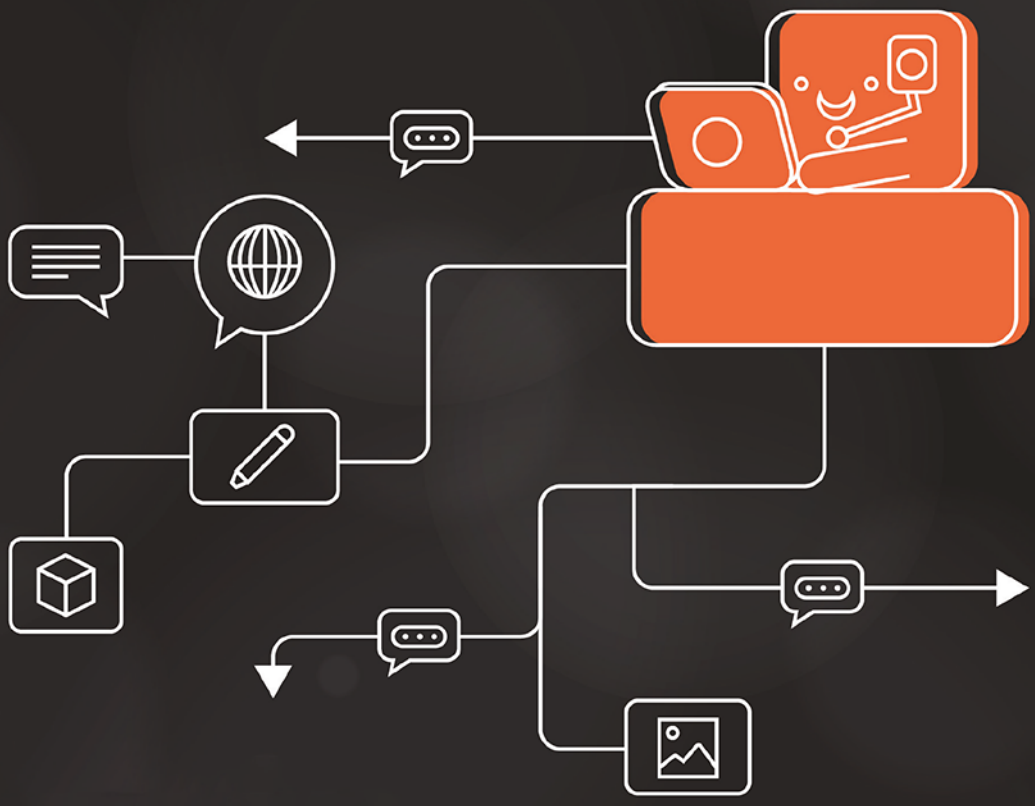
OpenAI API Receptury

Tworzenie inteligentnych aplikacji, chatbotów, wirtualnych asystentów i generatorów treści



Henry Habib

Helion

packt

Tytuł oryginału: OpenAI API Cookbook: Build intelligent applications including chatbots, virtual assistants, and content generators

Tłumaczenie: Grzegorz Werner

ISBN: 978-83-289-1758-3

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/openai
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

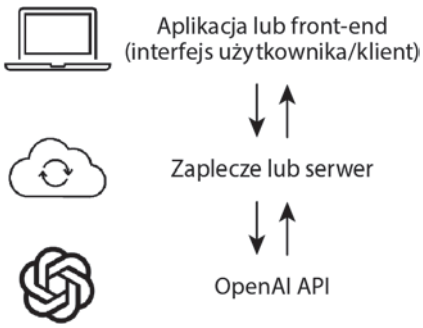
Spis treści

Rozdział 6

Budowanie inteligentnych aplikacji z użyciem OpenAI API

W tym rozdziale połączysz wszystkie kluczowe koncepcje, które poznałeś w poprzednich rozdziałach. Jest on poświęcony tworzeniu rzeczywistych inteligentnych aplikacji z użyciem OpenAI API. Trzeba pamiętać, że aplikacja to nie sam OpenAI API, ale również kilka otaczających go warstw, takich jak front-end i zaplecze.

Wykorzystasz tu architekturę aplikacji poznaną w poprzednim rozdziale. W szczególności użyjesz Google Cloud Functions jako warstwy zaplecza i Bubble jako warstwy front-endu. Jeśli nie pamiętasz ogólnej architektury aplikacji, zobacz jej poszczególne warstwy na rysunku 6.1.



Rysunek 6.1. Architektura typowej aplikacji używającej OpenAI API

W tym rozdziale omówimy następujące receptury:

- Tworzenie aplikacji nakładkowej, która generuje odpowiedzi na Twoje wiadomości e-mail.
- Tworzenie aplikacji multimodalnej, która generuje plan zwiedzania miasta.

Wymagania techniczne

Wszystkie receptury w tym rozdziale wymagają, żebyś miał dostęp do OpenAI API (za pośrednictwem wygenerowanego klucza API) i zainstalowanego klienta API. Więcej informacji o pozyskiwaniu klucza API znajdziesz w recepturze „Wykonywanie żądań OpenAI API za pomocą Postmana" w rozdziale 1. Musisz też wiedzieć, jak używać Pythona oraz biblioteki OpenAI Pythona, co zostało omówione w pierwszej recepturze w rozdziale 4.

Użyjesz też **Google Cloud Platform (GCP)** do hostingu swojego publicznego punktu końcowego. GCP to środowisko usług chmurowych udostępnianych przez Google. Oferuje ono szereg usług hostingowych i obliczeniowych związanych z bazami danych, przechowywaniem danych, analizą danych, uczeniem maszynowym i nie tylko, działających w infrastrukturze Google. Więcej informacji znajdziesz w recepturze „Tworzenie serwera publicznego punktu końcowego, który wywołuje OpenAI API" w rozdziale 5.

Musisz też wiedzieć, jak korzystać z Bubble, platformy programowania wizualnego, która pozwala użytkownikom tworzyć aplikacje bez pisania kodu. Więcej informacji na temat Bubble znajdziesz w recepturze „Wywołanie utworzonego punktu końcowego z aplikacji niewymagającej pisania kodu" w rozdziale 5.

Tworzenie aplikacji nakładkowej, która generuje odpowiedzi na Twoje wiadomości e-mail

W tej recepturze zbudujesz inteligentną aplikację, która pomaga Ci odpowiadać na wiadomości kierownika zlecającego Ci jakieś konkretne zadanie. Zawsze mam problem z grzecznym odmawianiem kierownikowi, kiedy jestem przeciążony pracą, więc bardzo doceniłbym taką aplikację (zresztą na co dzień korzystam z podobnego rozwiązania).

Z technicznego punktu widzenia wszystko, co robi ta aplikacja, można by zrobić poprzez bezpośrednie wywoływanie usługi ChatGPT. Po co więc poświęcać czas i budować ją w architekturze API, zaplecze, front-end? Kiedy uczysz się jakiejś nowej umiejętności, dobrze jest poznawać kolejne koncepcje w rozsądnym tempie. W tym przypadku pierwszym krokiem na drodze do zbudowania inteligentnej aplikacji jest zaczęcie od prostej aplikacji nakładkowej. Pozwoli Ci to opanować podstawowy proces budowania inteligentnej aplikacji. Następnie dodamy nowe koncepcje do tego procesu, co pozwoli Ci zbudować aplikację, której nie dałoby się utworzyć z wykorzystaniem samej usługi ChatGPT.

Aplikacja nakładkowa (ang. *wrapper application*) to zasadniczo warstwa oprogramowania, która pozwala na łatwiejszą i bardziej efektywną interakcję z OpenAI API. W programowaniu „**nakładka**" zwykle dotyczy oprogramowania, które działa jako pośrednik albo interfejs do innego komponentu lub interfejsu API i sprawia, że staje się on łatwiej dostępny albo prostszy w użyciu. Nakładki są bardzo użyteczne, ponieważ upraszczają interakcje z API; łatwiej też buduje się je w porównaniu z bardziej złożonymi aplikacjami multimodalnymi.

W tej recepturze zbudujesz aplikację w trzech fazach: OpenAI Playground, Google Cloud Function oraz Bubble.

Przygotowania

Upewnij się, że masz konto OpenAI Platform z dostępnym kredytem. Jeśli go nie masz, wykonaj czynności opisane w recepturze „Konfigurowanie środowiska OpenAI Playground" w rozdziale 1.

Upewnij się też, że utworzyłeś konto GCP. Aby utworzyć jakiekolwiek zasoby GCP, będziesz musiał określić profil rozliczeniowy. Zauważ, że GCP ma bezpłatną wersję próbną, a w tej recepturze *nie* przekroczymy jej limitów (więc zasadniczo nie zostaniesz obciążony żadnymi płatnościami).

Upewnij się wreszcie, że utworzyłeś konto Bubble. Bezpłatne konto możesz utworzyć pod adresem *http://bubble.io*.

Wymogi te dotyczą wszystkich receptur w tym rozdziale.

Jak to zrobić?

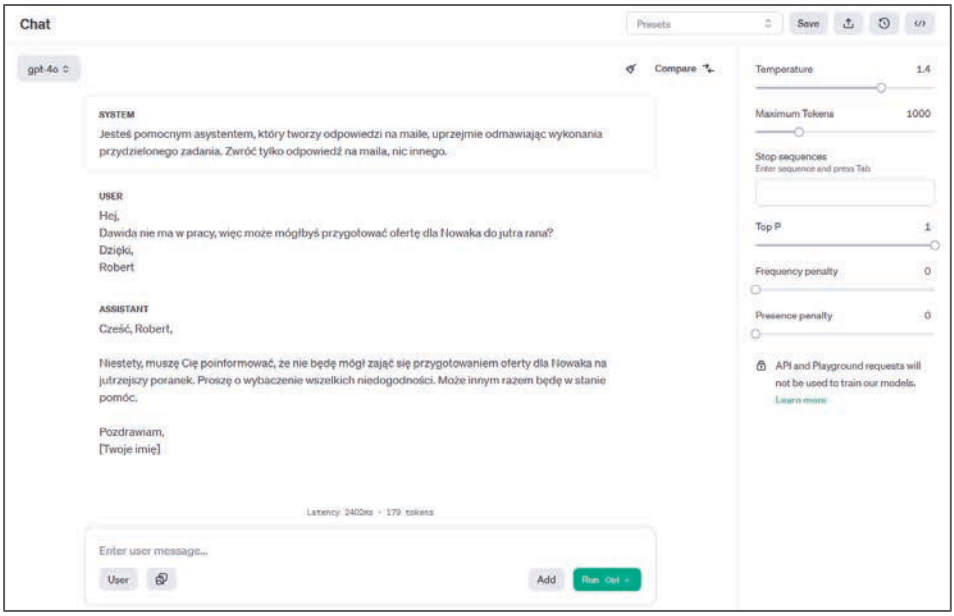
OpenAI Playground

1. Przejdź pod adres *https://platform.openai.com*. Z menu po lewej stronie wybierz tryb *Chat*.
2. W komunikacie systemowym wpisz **`Jesteś pomocnym asystentem, który tworzy odpowiedzi na maile, uprzejmie odmawiając wykonania przydzielonego zadania. Zwróć tylko odpowiedź na maila, nic innego`**.
3. W polu *User* kliknij *Enter a user message...* i wpisz przykładową wiadomość, taką jak poniższa:

```
Hej,
Dawida nie ma w pracy, więc może mógłbyś przygotować ofertę dla Nowaka do
jutra rana?
```

```
Dzięki,
Robert
```

4. Za pomocą przycisku w lewym górnym rogu zmień model z `gpt-3.5` na `gpt-4o`, a w menu po prawej stronie dokonaj następujących zmian we właściwościach środowiska Playground (rysunek 6.2):
   - Zwiększ parametr *Temperature* do `1.4`.
   - Ustaw parametr *Maximum Tokens* na około `1000`.



Rysunek 6.2. Konfiguracja środowiska Playground

5. Następnie kliknij przycisk *Run*. OpenAI utworzy dla Ciebie odpowiedź. W tym przypadku będzie to odpowiedź na przykładowego maila, uwzględniająca, że chcesz *uprzejmie* odmówić wykonania zadania, o które Cię poproszono.

Google Cloud Function

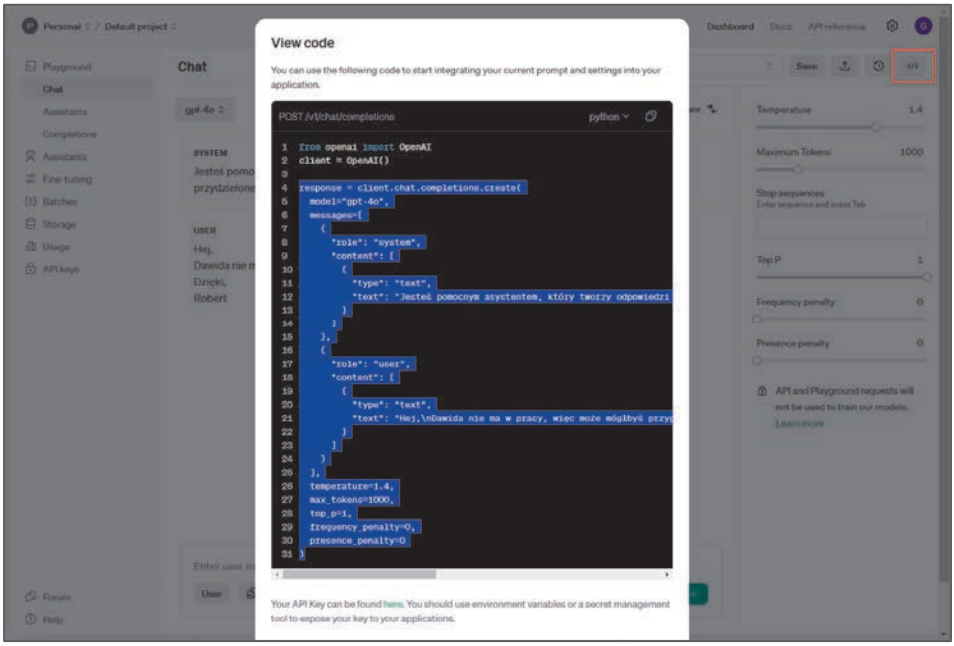
6. Na nowej karcie przejdź pod adres *https://cloud.google.com* i zaloguj się na swoim koncie Google.
7. Kliknij przycisk *Go to my console*.
8. Utwórz nową funkcję chmurową. W tym celu w polu wyszukiwania wpisz **`function`**, wybierz pozycję *Cloud Functions*, a następnie kliknij przycisk *Create Function*.

9. Nadaj funkcji opisową nazwę. Ponieważ ta funkcja będzie zwracała uprzejmą odpowiedź e-mail, nazwij ją `generatepoliteemail`.
10. W menu *Authentication* upewnij się, że wybrana jest opcja *Allow unauthenticated invocations*. Dzięki temu aplikacja frontendowa będzie mogła wywoływać warstwę zaplecza.
11. Kliknij przycisk *Next*, aby przejść do tworzenia funkcji. Z rozwijanego menu *Runtime* wybierz pozycję *Python 3.12*. W polu *Entry point* wpisz **`get_message`**.
12. W bloku kodu wpisz następujące instrukcje:

```
import functions_framework
from openai import OpenAI

@functions_framework.http
def get_message(request):

    request_json = request.get_json(silent=True) email =
    ↪request_json['email']
    client = OpenAI(api_key = '<Tutaj-Twój-klucz-API>')

    ### Tutaj kod Playground ###
    result = {
        'choice_1': response.choices[0].message.content,
        'choice_2': response.choices[1].message.content,
        'choice_3': response.choices[2].message.content,
    }

    return result
```

13. Przejdź do pliku *Requirements.txt* w menu po lewej stronie ekranu, dodaj nowy wiersz i wpisz **`openai`**. Dzięki temu na użytek tej funkcji zostanie pobrana biblioteka OpenAI.
14. Wróć do środowiska Open AI Playground. Usuń komunikat asystenta wygenerowany w etapie 4. Upewnij się, że wypełnione są tylko komunikat systemowy i komunikat użytkownika.
15. Kliknij znajdujący się w prawym górnym rogu przycisk *View code*, skopiuj funkcję `response` (rysunek 6.3) i w konsoli Google Cloud wklej ją do bloku kodu pod nagłówkiem `### Tutaj kod Playground ###` (rysunek 6.4).

    Kod ten wymaga pewnych zmian. Komunikat użytkownika jest obecnie statyczny (tzn. zawsze zaczyna się od `Hej, Dawida nie ma w pracy...`). Musisz zastąpić go zmienną `email`, która jest częścią tej funkcji.



Rysunek 6.3. Kod w środowisku Playground, który należy skopiować do funkcji chmurowej Google

Rysunek 6.4. Wklejanie kodu w konsoli Google Cloud

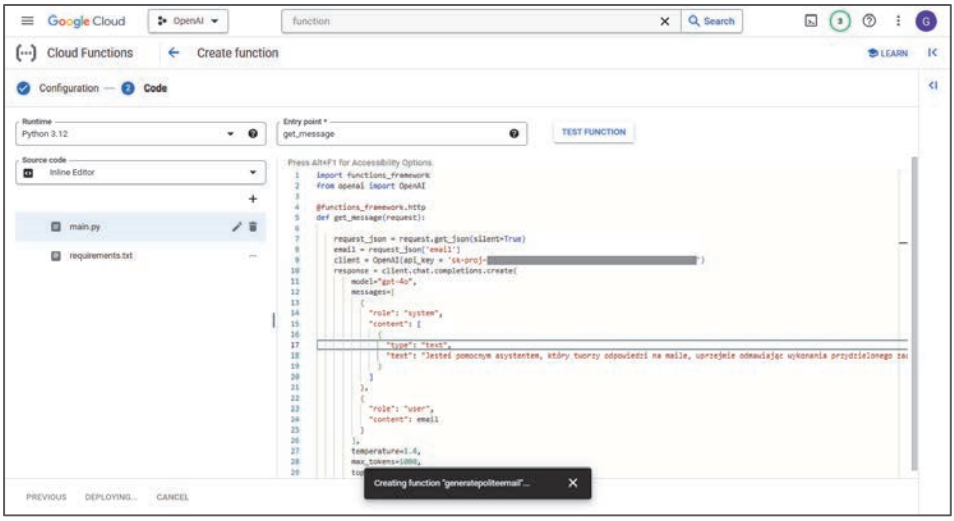
Ponadto, ponieważ chcesz otrzymywać trzy odpowiedzi e-mail (a zatem trzy odpowiedzi od OpenAI API), do żądania OpenAI musisz dodać argument `n=3`. Dzięki temu interfejs API będzie zwracał `n`, czyli trzy przykładowe wiadomości e-mail.

16. Po dokonaniu tych zmian blok kodu w funkcji chmurowej Google powinien wyglądać tak:

```
import functions_framework
from openai import OpenAI

@functions_framework.http
def get_message(request):

    request_json = request.get_json(silent=True)
    email = request_json['email']

    client = OpenAI(api_key = '<Tutaj-Twój-klucz-API>')

    response = client.chat.completions.create(
        model="gpt-4o",
        messages=[
          {
            "role": "system",
            "content": [
              {
                "type": "text",
                "text": "Jesteś pomocnym asystentem, który tworzy
                ↪odpowiedzi na maile, uprzejmie odmawiając wykonania
                ↪przydzielonego zadania. Zwróć tylko odpowiedź
                ↪na maila, nic innego."
            ]
          },
          {
            "role": "user",
            "content": email
          }
        ],
        temperature=1.4,
        max_tokens=1000,
        top_p=1,
        frequency_penalty=0,
        presence_penalty=0,
        n=3
    )

    result = {
        'choice_1': response.choices[0].message.content,
        'choice_2': response.choices[1].message.content,
```

```
        'choice_3': response.choices[2].message.content,
    }

    return result
```

17. Kliknij przycisk *Deploy*. Będziesz musiał zaczekać kilka minut na pełne wdrożenie funkcji. Kiedy zobaczysz zielony ptaszek na ekranie *Cloud Functions*, będzie to oznaczało, że Twoja funkcja została pomyślnie wdrożona (rysunek 6.5).

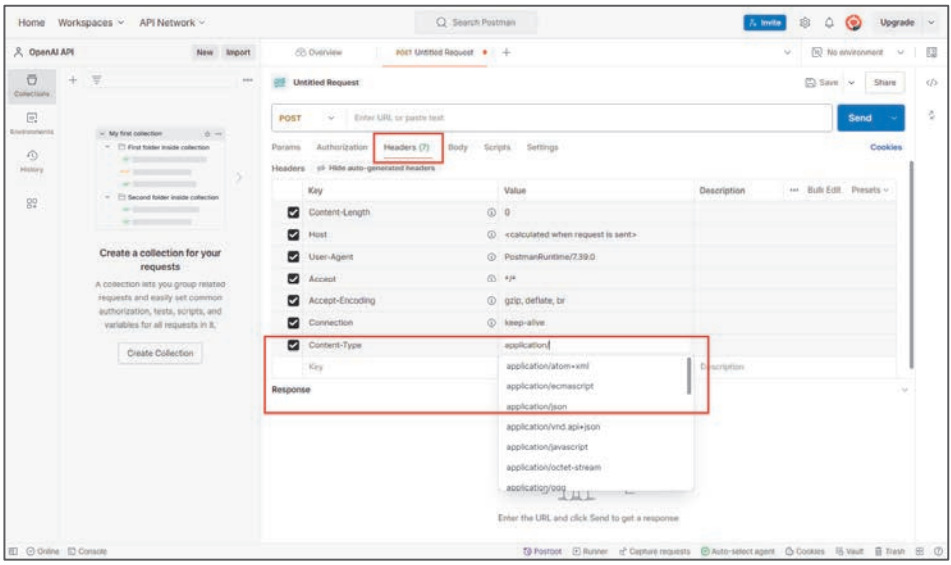


Rysunek 6.5. Wdrażanie funkcji chmurowej

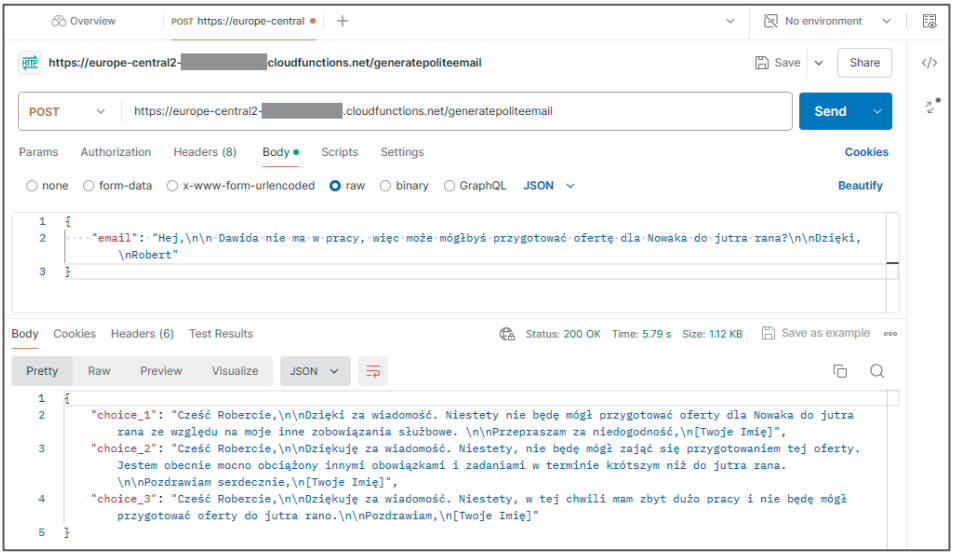
18. Teraz użyj Postmana, aby przetestować właśnie wdrożoną funkcję chmurową. W tym celu przejdź do obszaru roboczego Postmana, kliknij przycisk *New* znajdujący się w lewym górnym rogu i wybierz opcję *HTTP*.
19. W żądaniu Postmana wybierz sekcję *Headers* i wpisz nowy nagłówek, ustawiając *Key* na `Content-Type`, a *Value* na `application/json` (rysunek 6.6).
20. W menu rozwijanym zmień metodę żądania z *GET* na *POST*. Skopiuj adres URL punktu końcowego ze strony *Cloud Functions* i wklej go do Postmana.
21. Wybierz sekcję *Body*, a następnie zaznacz opcję *raw* i wklej poniższe żądanie JSON:

```
{
    "email": "Hej,\n\n Dawida nie ma w pracy, więc może mógłbyś
    ↪przygotować ofertę dla Nowaka do jutra rana?\n\nDzięki,\nRobert"
}
```

22. Kliknij przycisk *Send*, aby wywołać swoją funkcję chmurową. Jeśli wszystko pójdzie dobrze, powinieneś zobaczyć odpowiedź podobną do pokazanej na rysunku 6.7.



Rysunek 6.6. Konfiguracja testu funkcji chmurowej

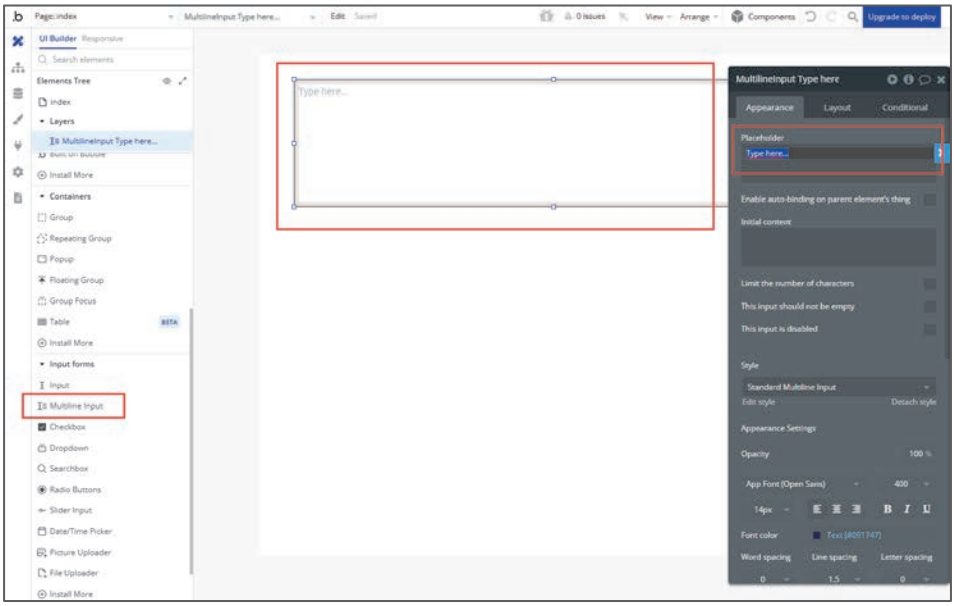


Rysunek 6.7. Udany test funkcji chmurowej

Bubble

23. Przejdź pod adres *http://bubble.io* i zaloguj się. Kliknij przycisk *Create an app* i nadaj swojej aplikacji odpowiednią nazwę. Kliknij przycisk *Get started* i wybierz opcję *Start with basic features*. Jeśli włączy się asystent tworzenia aplikacji, kliknij opcję *Skip application assistant*.

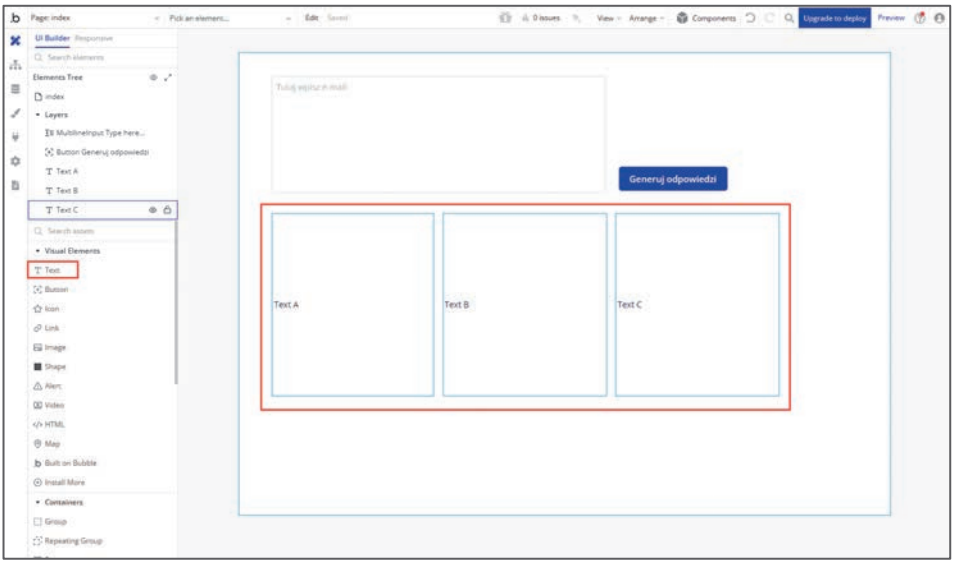
24. Na stronie kanwy dodasz kilka elementów potrzebnych do działania aplikacji. Wybierz element *Multiline Input* z listy po lewej stronie i narysuj prostokąt na górze strony. W menu właściwości elementu w polu *Placeholder* zastąp tekst *Type here...* tekstem `Tutaj wpisz e-mail:` (rysunek 6.8).



Rysunek 6.8. Konfigurowanie interfejsu użytkownika w Bubble.io

25. Utwórz element *Button* przez wybranie opcji *Button* i narysowanie prostokąta po prawej stronie pierwszego utworzonego elementu. Kliknij nowo utworzony element i w polu *Placeholder* zastąp tekst *...edit me...* tekstem `Generuj odpowiedzi`.

26. Ostatnim zbiorem elementów do utworzenia są trzy elementy *Text*. Wybierz opcję *Text* z menu po lewej stronie i narysuj pod elementem *Multiline Input* prostokąt na około 1/3 szerokości strony. Powtórz to dla pozostałych dwóch elementów *Text* i umieść je jeden obok drugiego (rysunek 6.9).



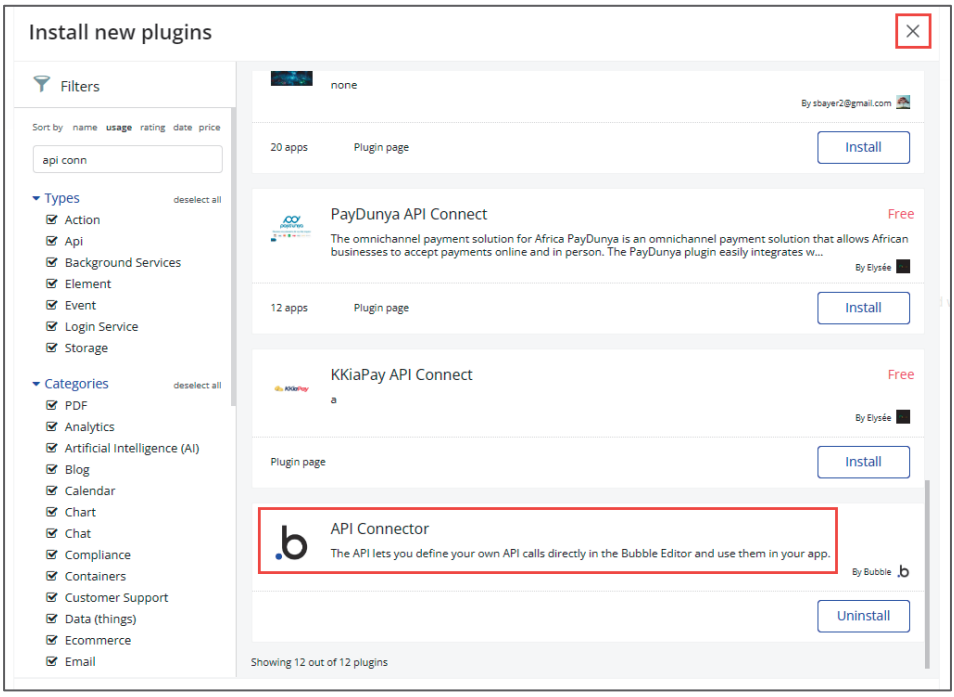
Rysunek 6.9. Konfigurowanie interfejsu użytkownika w Bubble.io

27. Kliknij element *Text A*, aby wyświetlić jego okno właściwości. Następnie kliknij przycisk *Insert dynamic data* przy polu tekstowym, wybierz pozycję *Text A*, a następnie wybierz opcję *Create a new custom state*. Zostaniesz zapytany o nazwę i typ stanu. W polu *State name* wpisz **`email_reply`**. W polu *State type* wybierz opcję *text*. Spowoduje to utworzenie unikatowego niestandardowego stanu, co jest wymagane do wyświetlenia wartości w aplikacji. Potwórz to samo dla elementów *Text B* i *Text C*.

28. Następnie musisz wczytać do Bubble utworzoną wcześniej funkcję chmurową. Z menu po lewej stronie wybierz opcję *Plugins* (ikonę wtyczki) i kliknij przycisk *Add Plugins*. Zaznacz opcję *API Connector*, kliknij przycisk *Install*, a następnie kliknij przycisk × w prawym górnym rogu (rysunek 6.10).

29. Zaznacz API Connector na liście wtyczek, a następnie kliknij przycisk *Add Another API*. W polu *API name* wpisz **`get_replies`**. Przewiń okno w dół, aż dotrzesz do pola *Name*, i kliknij łącze *expand*. Zmień nazwę tego wywołania API na `get_replies`. Ustawienia API skonfiguruj w następujący sposób:

    - Z listy rozwijanej *Use as* wybierz opcję *Action*.
    - Zmień metodę żądania z *GET* na *POST*.
    - Tuż obok pola *POST* jest miejsce na wprowadzenie adresu URL punktu końcowego. Wprowadź adres URL skopiowany z funkcji chmurowej Google.
    - Utwórz nowy nagłówek przez kliknięcie przycisku *Add Header*. W polu *Key* wpisz **`Content-Type`**, a w polu *Value* — **`application/json`**.

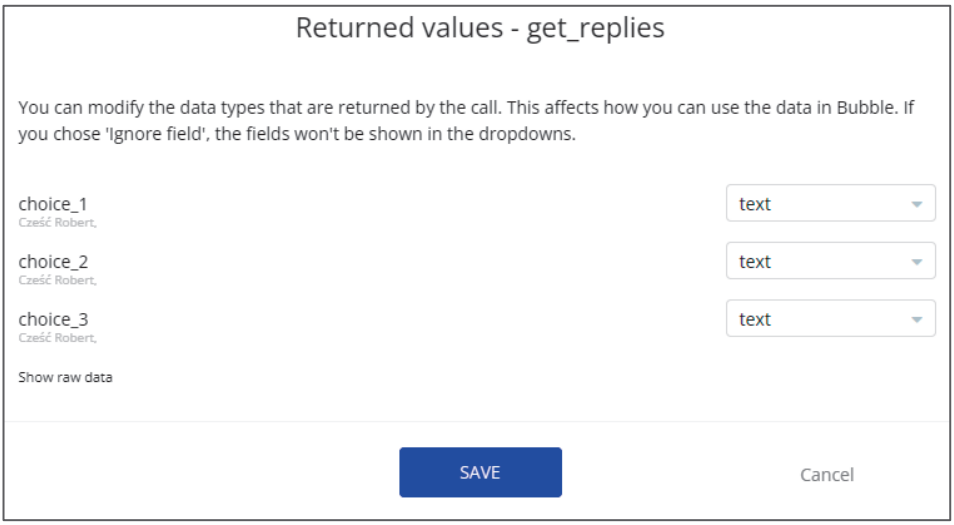


Rysunek 6.10. Konfigurowanie interfejsu użytkownika w Bubble.io

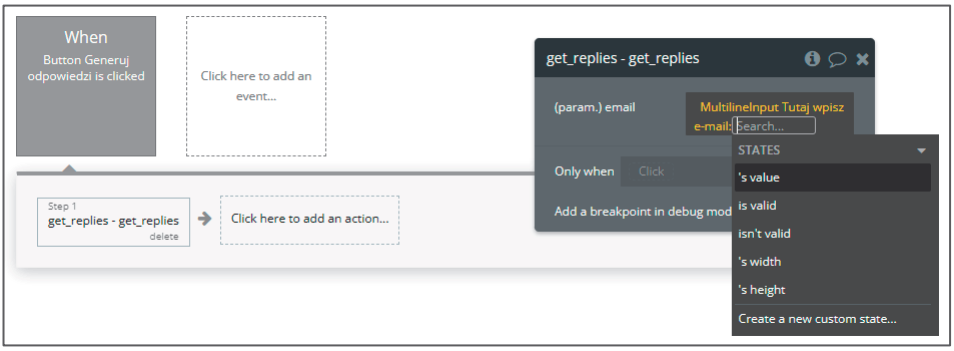
- Utwórz nowy parametr przez kliknięcie przycisku *Add parameter*. W polu *Key* wpisz **`email`**. W polu *Value* wprowadź ten sam tekst JSON, którego użyłeś w wywołaniu Postmana w etapie 21.; zamieszczono go również poniżej. Nie dołączaj znaków cudzysłowu. Upewnij się, że pole wyboru *Private nie jest* zaznaczone. Jest to ważne, ponieważ dzięki temu parametr ten będzie dostępny w aplikacji Bubble:

  ```
  Hej,\n\n Dawida nie ma w pracy, więc może mógłbyś przygotować ofertę
  dla Nowaka do jutra rana?\n\nDzięki,\nRobert
  ```

30. Kliknij przycisk *Initialize Call*, aby przetestować wywołanie API. Jeśli zobaczysz ekran pokazany na rysunku 6.11, to wywołanie przebiegło pomyślnie. Upewnij się, że dla każdego pola *choice* wybrany jest typ *text*, i kliknij przycisk *Save*.
31. Wybierz ikonę *Design* z menu po lewej stronie. Kliknij utworzony wcześniej element *Button*. W wyświetlonym menu właściwości kliknij przycisk *Add Workflow*.
32. Kliknij przycisk *Click here to add an action*. Wskaż opcję *Plugins*, znajdź właśnie utworzone wywołanie API (`get_replies - get_replies`) i zaznacz je. W wyświetlonym menu właściwości usuń zawartość pola *(param.) email* i kliknij przycisk *Insert dynamic data*. Przewiń listę w dół, wybierz pozycję *Multiline Input Tutaj wpisz e-mail:.*, a następnie wybierz pozycję *value* (rysunek 6.12).



Rysunek 6.11. Pomyślny wynik inicjalizacji wywołania API



Rysunek 6.12. Proces Bubble

33. Teraz ponownie kliknij przycisk *Click here to add an action*, przewiń listę w dół do pozycji *Element Actions* i wybierz opcję *Set State*. Z listy rozwijanej *Element* wybierz pozycję *Text A*. Z listy rozwijanej *Custom state* wybierz pozycję `email_reply`. W polu *Value* wybierz opcję *Results of step 1*, a następnie *choice_1*. Spowoduje to ustawienie wartości elementu *Text A* na pierwszy wybór w wynikach wywołania utworzonej wcześniej funkcji chmurowej.
34. Powtórz etap 33. kolejno dla elementów *Text B* i *Text C*, wybierając odpowiednio pozycje *choice_2* i *choice_3*.
35. Twoja inteligentna aplikacja Bubble jest gotowa. Aby sprawdzić, czy działa, kliknij przycisk *Preview* znajdujący się w prawym górnym rogu okna; pojawi się nowa strona z Twoją aplikacją. W polu tekstowym *Tutaj wpisz e-mail* spróbuj

wpisać tę samą wiadomość e-mail, której użyłeś w środowisku OpenAI Playground w etapie 3.:

```
Hej,
Dawida nie ma w pracy, więc może mógłbyś przygotować ofertę dla Nowaka do
jutra rana?
Dzięki,
Robert
```

36. Kliknij przycisk *Generuj odpowiedzi*. Jeśli wszystko pójdzie dobrze, powinieneś zobaczyć ekran podobny do pokazanego na rysunku 6.13, który pokazuje trzy możliwe odpowiedzi uprzejmie odmawiające wykonania zadania.

Rysunek 6.13. Pomyślnie skonfigurowana aplikacja

Jak to działa?

W tej recepturze utworzyłeś prostą nakładkową aplikację webową, która generuje przykładowe odpowiedzi na wiadomości e-mail. Dzięki użyciu OpenAI aplikacja ta wykorzystuje potęgę modeli LLM do rozwiązania konkretnego problemu. W tym przypadku chodzi o znalezienie sposobu na uprzejmą odmowę wykonania zadania przydzielonego w mailu.

Poszczególne etapy tej procedury przypominały czynności, które znasz z poprzednich receptur:

1. Najpierw przetestowałeś podpowiedzi w OpenAI Playground, środowisku, w którym użytkownicy mogą wypróbowywać różne konfiguracje i obserwować ich wpływ na generowane odpowiedzi.
2. Następnie utworzyłeś funkcję chmurową Google, która jest warstwą zaplecza. W warstwie tej dodałeś kod ze środowiska Playground, który wywołuje OpenAI API.
3. Następnie przetestowałeś swoją warstwę zaplecza z użyciem Postmana, aby upewnić się, że wywołania działają poprawnie i że otrzymujesz odpowiednie odpowiedzi od warstwy zaplecza.
4. Następnie utworzyłeś prosty front-end z użyciem Bubble.
5. Na koniec połączyłeś warstwy front-endu i zaplecza z wykorzystaniem procesów Bubble i wtyczki API Connector.

Wykonując te czynności, możesz utworzyć dowolną inteligentną aplikację w niespełna godzinę. Warto przyjrzeć się bliżej etapowi 1., ponieważ jest to krytyczny krok w tworzeniu każdej nowej aplikacji.

Zacząłeś tę recepturę w środowisku OpenAI Playground, gdzie przetestowałeś **komunikat systemowy** w środowisku, które pozwala na szybkie iteracje, aby upewnić się, że otrzymujesz właściwe odpowiedzi. Środowisko Playground ma przyjazny dla użytkownika interfejs, który pomaga eksperymentować z różnymi podpowiedziami i parametrami. Eksperymenty te mają kluczowe znaczenie dla zrozumienia, jak model AI reaguje na różne dane wejściowe, co z kolei pomaga dostosować odpowiedzi do potrzeb tworzonej aplikacji.

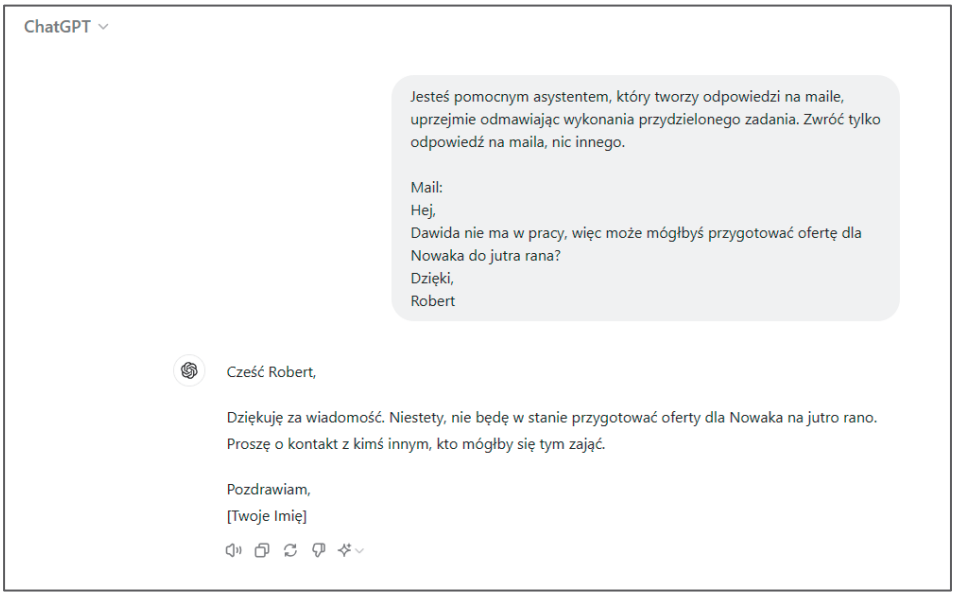
W środowisku Playground i w skrypcie Pythona wykonującym żądanie OpenAI API ustawiłeś następujące parametry:

- **Model** na `gpt-4`. W przypadku aplikacji odpowiadającej na wiadomości e-mail zaawansowane rozumienie kontekstu i niuansów ma kluczowe znaczenie. Wiadomości mogą mieć szeroką gamę tematów i stylów, od formalnej komunikacji biznesowej do niezobowiązujących konwersacji. Zaawansowany model językowy GPT-4 potrafi adaptować się do tych różnorodnych stylów i dostarczać odpowiedzi, które są dokładniejsze i lepiej pasują do kontekstu.
- **Temperature** na `1.4`. Ponieważ celem tej aplikacji jest tworzenie kilku zróżnicowanych odpowiedzi, temperatura została ustawiona na wyższą od standardowej (`1.0`), żeby zagwarantować pewną losowość odpowiedzi. Wyższa temperatura pomaga w generowaniu odpowiedzi, które nie są szablonowe i mogą adaptować się do unikatowej treści i tonu każdej wiadomości e-mail.

- **Maximum Tokens** na 1000. Maksymalna liczba tokenów określa, jak długie mogą być generowane odpowiedzi. Ustawienie tego parametru na 1000 pozwala uzyskać wystarczająco długie odpowiedzi.
- **N** na 3. Parametr *N* określa liczbę odpowiedzi generowanych przez model dla każdej podpowiedzi. Ustawienie *N* na 3 oznacza, że aplikacja będzie generować trzy różne odpowiedzi na każdą wiadomość e-mail.

Inną zaletą środowiska Playground jest to, że generuje ono gotowy do użycia kod Pythona; wykorzystałeś to w etapie 15. niniejszej receptury. Kiedy skonfigurowałeś wszystkie ustawienia w środowisku Playground i byłeś zadowolony z odpowiedzi, wystarczyło kliknąć przycisk *View Code*, aby wygenerować wymagany kod.

W tej recepturze utworzyłeś prostą aplikację. Ponieważ jednak była to aplikacja nakładkowa, ten sam efekt można osiągnąć w samej usłudze ChatGPT, bez tworzenia złożonych warstw zaplecza i front-endu. Mogę na przykład przejść na stronę ChatGPT (*https://chat.openai.com/*) i wpisać poniższą podpowiedź, na którą otrzymam podobne odpowiedzi (rysunek 6.14).



Rysunek 6.14. Tworzenie odpowiedzi na maile z użyciem ChatGPT

Może więc zastanawiasz się, jaki jest sens tworzenia aplikacji z użyciem OpenAI API? Cóż, niektóre aplikacje nie są prostymi nakładkami i można je utworzyć tylko z wykorzystaniem API. Właśnie to omówimy w następnej recepturze.

Tworzenie aplikacji multimodalnej, która generuje plan zwiedzania miasta

W poprzedniej recepturze utworzyłeś inteligentną aplikację, która generowała odpowiedzi na wiadomości e-mail. Dowiedziałeś się też, że z technicznego punktu widzenia była to aplikacja nakładkowa, czyli że ten sam efekt można łatwo osiągnąć za pomocą samej usługi ChatGPT albo środowiska Playground.

W tej recepturze zrobisz następny krok i utworzysz aplikację multimodalną. **Aplikacja multimodalna** to złożony program, które integruje różne typy mediów i metody interakcji w jedną *spójną* całość. Dzięki tej integracji interfejs użytkownika jest bogatszy i bardziej zajmujący, przez co może spełnić szerszą gamę preferencji i potrzeb użytkowników.

Kluczowym aspektem aplikacji multimodalnej jest łączenie tekstu, głosu, obrazów, a może nawet wideo w celu stworzenia bardziej dynamicznego i interaktywnego środowiska. Rozważ na przykład aplikację, która nie tylko odpowiada na kwerendy tekstowe, ale również rozumie polecenia głosowe, potrafi analizować obrazy, a w odpowiedzi może nawet generować treść wideo. Taka aplikacja byłaby niezwykle przydatna w takich dziedzinach, jak edukacja, w której wspierałaby różne style uczenia się, albo obsługa klienta, w której mogłaby zapewnić bardziej spersonalizowaną i efektywną obsługę.

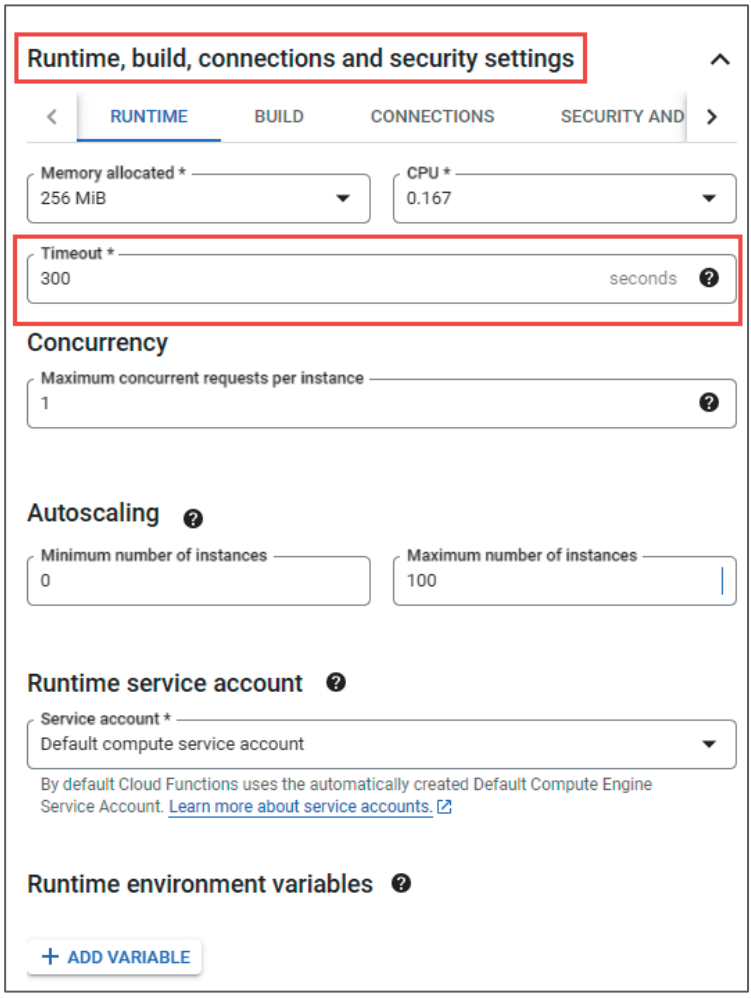
W tym przykładzie połączysz **Chat API** i **Images API** w celu tworzenia planów zwiedzania miasta. Użytkownik może przejść do aplikacji, wpisać nazwę miasta, do którego się udaje, i otrzymać całodzienny plan zwiedzania ze zdjęciami obiektów wspomnianych w utworzonym planie.

Jest to przykład aplikacji, której *nie* dałoby się łatwo utworzyć z użyciem samej usługi ChatGPT lub środowiska Playground, a więc takiej, która zapewnia rzeczywistą korzyść.

Jak to zrobić?

1. Na nowej karcie przeglądarki przejdź pod adres *https://cloud.google.com* i zaloguj się na swoim koncie Google.
2. Kliknij przycisk *Console* znajdujący się w prawym górnym rogu okna.
3. Utwórz nową funkcję chmurową Google. W pasku wyszukiwania wpisz **`function`**, wybierz z listy pozycję *Cloud Functions* i kliknij przycisk *Create Function*.
4. Nadaj funkcji opisową nazwę. Ponieważ funkcja będzie tworzyć plan zwiedzania wraz z obrazami, nazwij ją `get_itinerary_and_images`.

5. W menu *Authentication* upewnij się, że zaznaczona jest opcja *Allow unauthenticated invocations*. Dzięki temu aplikacja frontendowa będzie mogła wywoływać warstwę zaplecza.
6. Kliknij pozycję *Runtime, build, connections and security settings*, aby rozwinąć opcje. Zmień ustawienie *Timeout* z 60 sekund na 300 sekund. Dzięki temu limit czasu działania funkcji chmurowej wydłuży się z 1 minuty do 5 minut. Jest to ważne w aplikacji multimodalnej, ponieważ będzie ona wykonywać kilka żądań API (rysunek 6.15).



Rysunek 6.15. Ustawienia konfiguracyjne funkcji chmurowej Google

7. Kliknij przycisk *Next*, aby przejść do tworzenia funkcji. Z rozwijanego menu *Runtime* wybierz pozycję *Python 3.12*. W polu *Entry point* wpisz **`get_travel_details`**.
8. Przejdź do pliku *Requirements.txt* w menu po lewej stronie ekranu, dodaj nowy wiersz i wpisz **`openai`**. Dzięki temu na użytek tej funkcji zostanie pobrana biblioteka OpenAI.
9. W *bloku kodu* wpisz poniższe instrukcje. Ta funkcja przyjmuje nazwę miasta i zwraca plan zwiedzania w formacie rano – po południu – wieczorem oraz trzy odpowiednie obrazy, po jednym dla poranka, popołudnia i wieczoru.

```
import functions_framework
from openai import OpenAI

@functions_framework.http
def get_travel_details(request):

    request_json = request.get_json(silent=True)
    city = request_json['city']

    client = OpenAI(api_key = '<tutaj klucz API>')

    response = client.chat.completions.create(
      model="gpt-4o",
      messages=[
    {
      "role": "system",
      "content": "Jesteś pomocnym asystentem, który tworzy jednodniowe
      ↪plany zwiedzania miasta wybranego przez użytkownika. Utwórz tylko
      ↪3 aktywności (rano, po południu, wieczór). Podaj tylko plan
      ↪zwiedzania, nic innego."
    },
    {
      "role": "user",
      "content": "Rzym, Włochy"
    },
    {
      "role": "assistant",
      "content": "Rano: \n\Rozpocznij dzień w Koloseum, jednym
      ↪z najbardziej znanych zabytków Rzymu. Wybierz się na wycieczkę
      ↪z przewodnikiem, aby w pełni docenić jego historię i znaczenie.
      ↪\n\nPopołudnie: \n\nUdaj się do Watykanu. Odwiedź Muzea
      ↪Watykańskie, w których znajduje się ogromna kolekcja dzieł sztuki
      ↪i artefaktów historycznych. Nie przegap Kaplicy Sykstyńskiej,
      ↪słynącej z sufitu z freskiem Michała Anioła.\n\ nWieczór:
      ↪\n\nWybierz się na spokojny spacer do fontanny di Trevi.
      ↪Pamiętaj, aby na szczęście wrzucić do fontanny monetę przez
      ↪ramię. Następnie zakończ dzień pyszną włoską kolacją w jednej
      ↪z pobliskich restauracji."
```

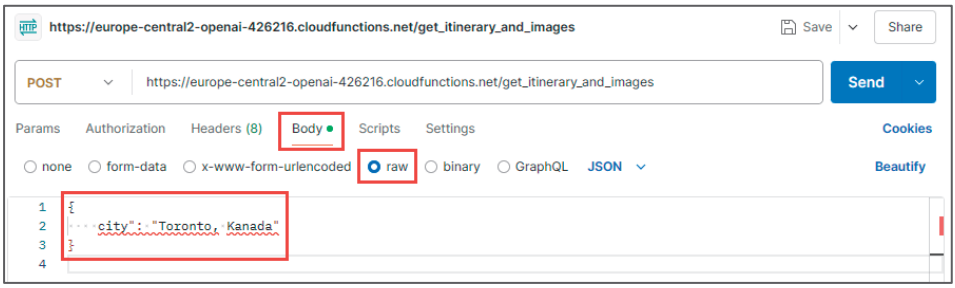
```
},
{
  "role": "user",
  "content": "Lizbona, Portugalia"
},
{
  "role": "assistant",
  "content": "Rano: \n\nRozpocznij dzień wizytą w słynnej wieży
  ↪Belem, wpisanej na Listę Światowego Dziedzictwa UNESCO, skąd
  ↪roztacza się panoramiczny widok na Lizbonę. \n\nPopołudnie:
  ↪\n\nOdkryj historyczną dzielnicę Alfama. Przechadzaj się wąskimi,
  ↪krętymi uliczkami, odwiedź katedrę Se i zjedz tradycyjny
  ↪portugalski lunch w lokalnej tawernie.\n\nWieczór: \n\nUdaj się
  ↪na kolację do Bairro Alto, artystycznej dzielnicy miasta.
  ↪Następnie wybierz się na pokaz fado - tradycyjnego portugalskiego
  ↪tańca - w jednym z lokalnych barów."
},
{
  "role": "user",
  "content": city
},
],
  temperature=0.64,
  max_tokens=1024,
  top_p=1,
  frequency_penalty=0,
  presence_penalty=0
)

itinerary = response.choices[0].message.content

response = client.chat.completions.create(
  model="gpt-4o",
  messages=[
    {
      "role": "system",
      "content": "Jesteś pomocnym asystentem, który tworzy
      ↪podpowiedzi dla DALL-E na podstawie planu zwiedzania.
      ↪Podpowiedzi powinny być krótkie. Utwórz jedną podpowiedź
      ↪dotyczącą poranka, jedną dotyczącą popołudnia i jedną
      ↪dotyczącą wieczora. Podpowiedzi dla DALL-E powinny być
      ↪rozdzielone znakiem \"|\"."
    },
    {
      "role": "user",
      "content": itinerary
    }
  ],
  temperature=0.64,
  max_tokens=1024,
```

```
    top_p=1,
    frequency_penalty=0,
    presence_penalty=0
)

dalle_prompts = response.choices[0].message.content
dalle_prompts_list = response.choices[0].message.content.split('|')

image_urls =  []
for prompt in dalle_prompts_list:
    response = client.images.generate(
          model="dall-e-3",
          prompt=prompt,
          size="1024x1024",
          quality="standard",
          n=1
    )
    image_urls.append(response.data[0].url)
result = {
    'itinerary': itinerary,
    'morning_image': image_urls[0],
    'afternoon_image': image_urls[1],
    'evening_image': image_urls[2]
}
return result
```

10. Kliknij przycisk *Deploy*. Będziesz musiał zaczekać kilka minut na pełne wdrożenie funkcji. Kiedy zobaczysz zielony ptaszek na ekranie *Cloud Functions*, będzie to oznaczało, że Twoja funkcja została pomyślnie wdrożona.
11. Podobnie jak w poprzedniej recepturze, użyjesz Postmana do przetestowania właśnie wdrożonej funkcji. Przejdź na stronę Postmana, kliknij przycisk *New* znajdujący się w lewym górnym rogu i wybierz metodę *HTTP*.
12. W żądaniu Postmana wybierz sekcję *Headers* i utwórz nowy nagłówek z polem *Key* ustawionym na `Content-Type` i polem *Value* ustawionym na `application/json`.
13. Zmień typ żądania z *Get* na *Post*. Skopiuj URL punktu końcowego ze strony funkcji chmurowej i wklej go do Postmana.
14. Wybierz sekcję *Body*, zaznacz opcję *raw*, a następnie skopiuj i wklej poniższe żądanie JSON (rysunek 6.16).

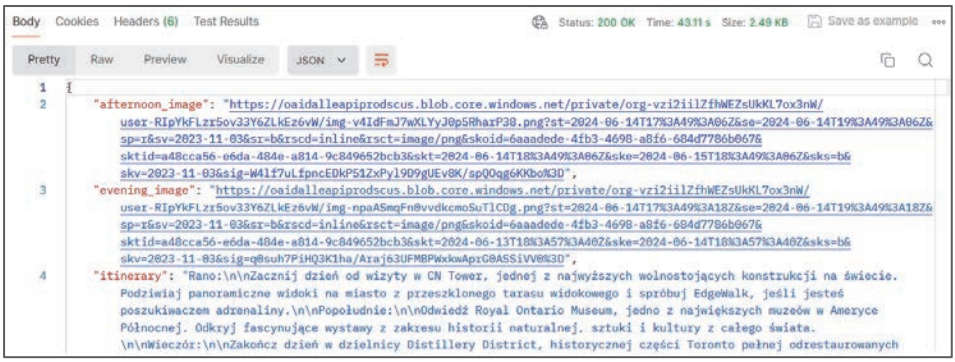
```
{
    "city": "Toronto, Kanada"
}
```

Zauważ, że jest to długa funkcja chmurowa, która wysyła kilka żądań do OpenAI (dotyczących zarówno tekstu, jak i obrazów), więc jej wykonanie może zająć kilka minut.



Rysunek 6.16. Sekcja Body w Postmanie

15. Kliknij przycisk *Send*, aby wywołać swoją funkcję chmurową. Jeśli wszystko pójdzie dobrze, powinieneś zobaczyć odpowiedź podobną do pokazanej na rysunku 6.17, która zawiera kilka obiektów osadzonych w obiekcie JSON:



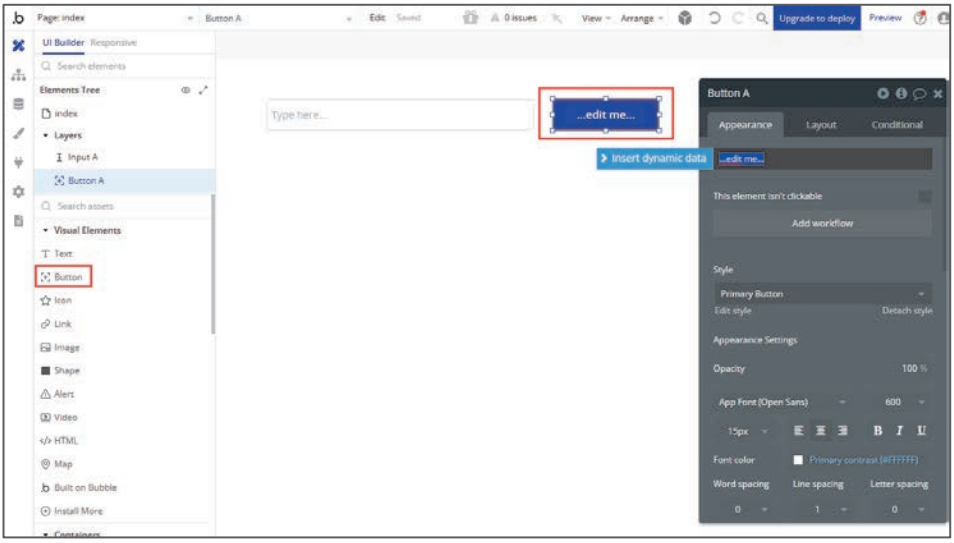
Rysunek 6.17. Wyniki wywołania w Postmanie

16. Przejdź pod adres *http://bubble.io* i zaloguj się. Kliknij przycisk *Create an App* i nadaj swojej aplikacji odpowiednią nazwę. Kliknij przycisk *Get Started* i wybierz opcję *Start with Basic Features*. Na następnym ekranie kliknij przycisk *Skip the application assistant*.

17. Teraz na stronie **kanwy** dodasz kilka elementów potrzebnych do działania aplikacji. Dodaj pole wejściowe przez wybranie elementu *Input* z menu po lewej stronie i narysowanie prostokąta na górze strony. Następnie dodaj przycisk przez wybranie elementu *Button* z menu po lewej stronie i narysowanie przycisku obok elementu *Input* (rysunek 6.18).

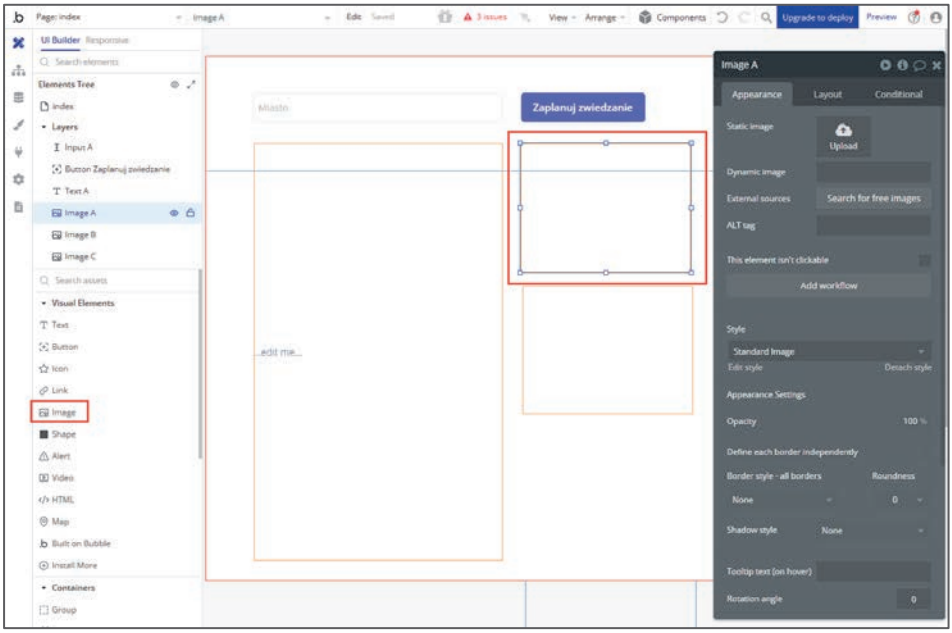
18. Kliknij dwukrotnie element *Input* i w menu właściwości zastąp tekst w polu *Placeholder* przez `Miasto`.

19. Kliknij dwukrotnie element *Button* i w menu właściwości zastąp tekst *...edit me...* przez `Zaplanuj zwiedzanie`.



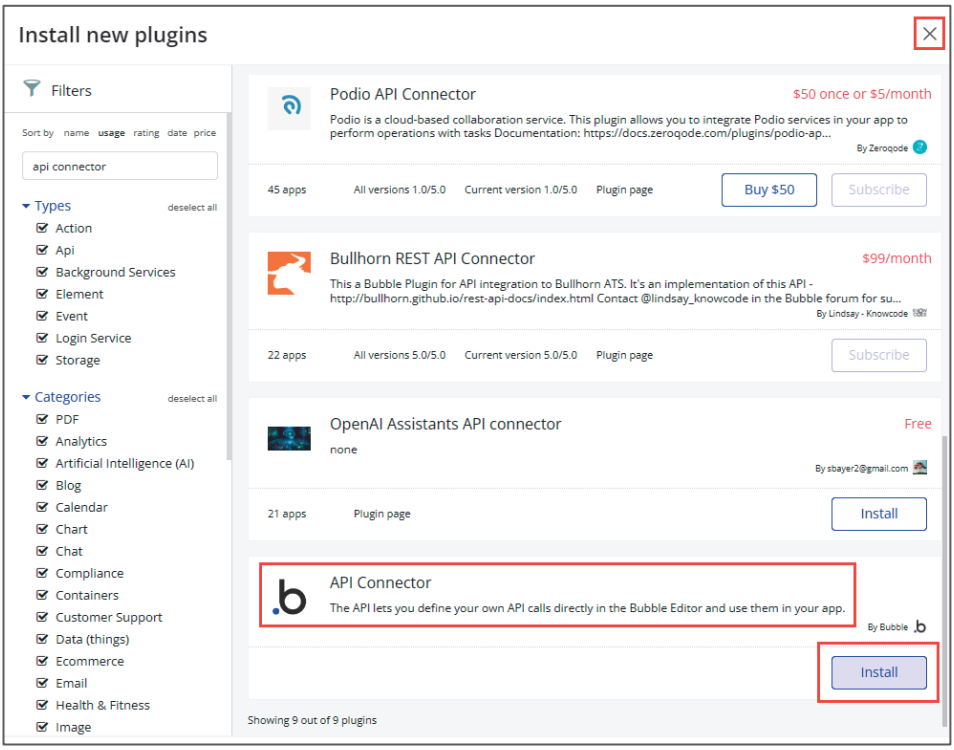
Rysunek 6.18. Dodawanie przycisku w Bubble

20. Utwórz element *Text* i umieść go po lewej stronie ekranu.
21. Teraz musisz utworzyć trzy obrazy. Wybierz element *Image* i umieść go po prawej stronie ekranu. Zrób to trzykrotnie, tworząc elementy *Image A*, *Image B* i *Image C*. Upewnij się, że mają one mniej więcej ten sam rozmiar; może będziesz musiał w tym celu je przeskalować i poprzesuwać (rysunek 6.19).
22. Następnie utwórz nowe stany niestandardowe dla elementów *Text* i *Image*.
23. Kliknij element *Text*, aby wyświetlić jego menu właściwości. Następnie kliknij przycisk *Insert dynamic data* w polu tekstowym, wybierz pozycję *Text*, a następnie *Create a new custom state*. Zostaniesz zapytany o nazwę i typ stanu. W polu *State name* wpisz **`itinerary_details`**. Z listy rozwijanej *State type* wybierz pozycję *text*. Spowoduje to utworzenie unikatowego niestandardowego stanu dla pola tekstowego, co jest wymagane do wyświetlania wartości w aplikacji. Pole to będzie zawierać plan zwiedzania utworzony przez aplikację.
24. Dla każdego elementu *Image* kliknij element, aby wyświetlić jego menu właściwości. Następnie kliknij przycisk *Insert dynamic data* w polu *Dynamic image*. Wybierz pozycję *Image X*, a następnie *Create a new custom state*. Zostaniesz zapytany o nazwę i typ stanu. W polu *State name* wpisz **`img_url`**. Z listy rozwijanej *State type* wybierz pozycję *text*.
25. Następnie musisz wczytać do Bubble utworzoną wcześniej funkcję chmurową. Wybierz pozycję *Plugins* (ikonę wtyczki) z menu po lewej stronie i kliknij przycisk *Add Plugins*. Wybierz wtyczkę *API Connector*, kliknij przycisk *Install*, a następnie kliknij × w lewym górnym rogu (rysunek 6.20).



Rysunek 6.19. Dodawanie obrazów w Bubble

26. Kliknij przycisk *Add Another API*. W polu *API Name* wpisz **`travel`**. Przewiń okno w dół do pola *Name* i kliknij łącze *expand*. Zmień nazwę tego wywołania API na `travel`. Wprowadź następujące ustawienia wywołania API:
    - Z listy rozwijanej *Use as* wybierz opcję *Action*.
    - Zmień metodę żądania z *GET* na *POST*.
    - Tuż obok pola *POST* jest miejsce na wprowadzenie *adresu URL punktu końcowego*. Wprowadź adres URL skopiowany z funkcji chmurowej Google.
    - Utwórz nowy nagłówek przez kliknięcie przycisku *Add Header*. W polu *Key* wpisz **`Content-Type`**, a w polu *Value* — **`application/json`**.
    - Utwórz nowy parametr przez kliknięcie przycisku *Add parameter*. W polu *Key* wpisz **`city`**. W polu *Value* wprowadź ten sam tekst JSON, którego użyłeś w wywołaniu Postmana, czyli `Toronto, Kanada`. Upewnij się, że pole wyboru *Private nie jest* zaznaczone (rysunek 6.21).
27. Kliknij przycisk *Initialize Call*, aby przetestować wywołanie API. Jeśli zobaczysz ekran pokazany na rysunku 6.22, to wywołanie przebiegło pomyślnie. Upewnij się, że dla każdego zwróconego obiektu wybrany jest typ *text*, i kliknij przycisk *Save*.

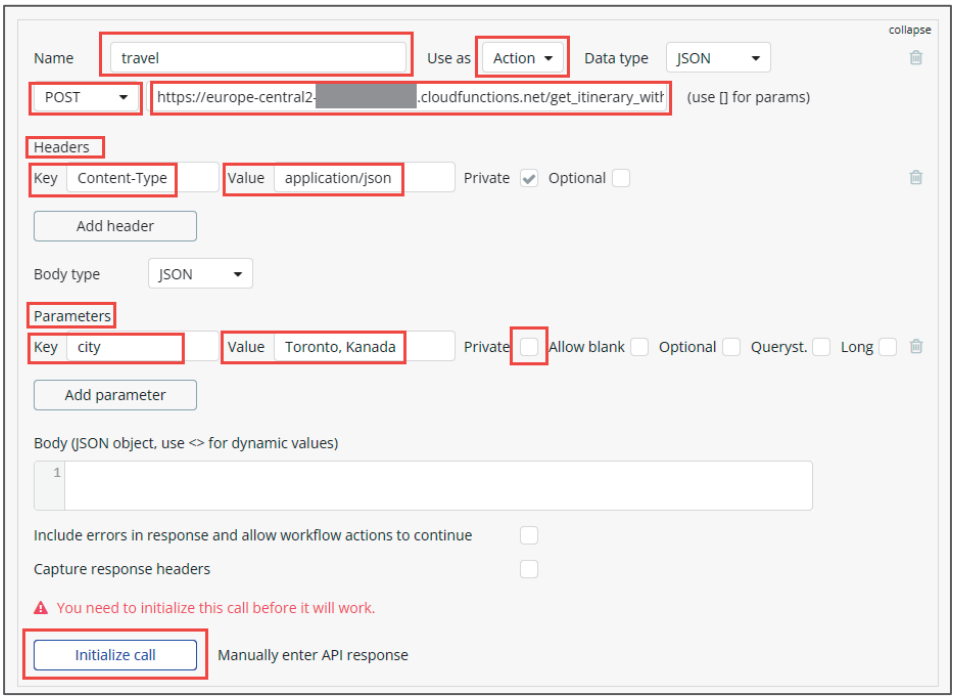


Rysunek 6.20. Konfigurowanie interfejsu użytkownika w Bubble.io

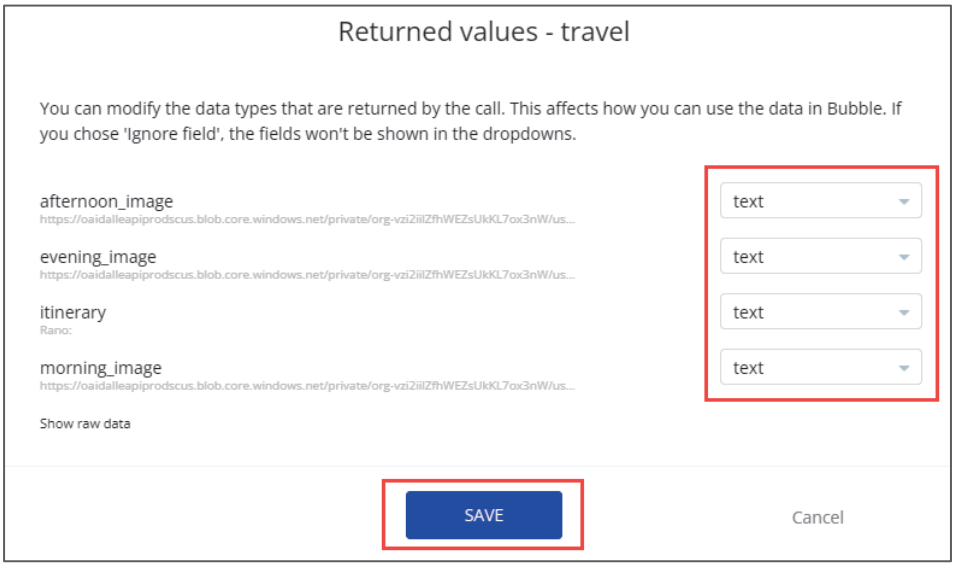
28. Wybierz ikonę *Design* z menu po lewej stronie. Kliknij utworzony wcześniej element *Button*. W wyświetlonym menu właściwości kliknij przycisk *Add Workflow*.

29. Kliknij przycisk *Click here to add an action*. Następnie przejdź do pozycji *Plugins*, znajdź utworzone przed chwilą wywołanie API (`travel – travel`) i je zaznacz. W wyświetlonym menu właściwości usuń zawartość pola *(param.) city* i kliknij przycisk *Insert dynamic data*. Przewiń listę w dół, wybierz opcję *Input Miasto*, a następnie wybierz opcję *value* (rysunki 6.23 i 6.24).

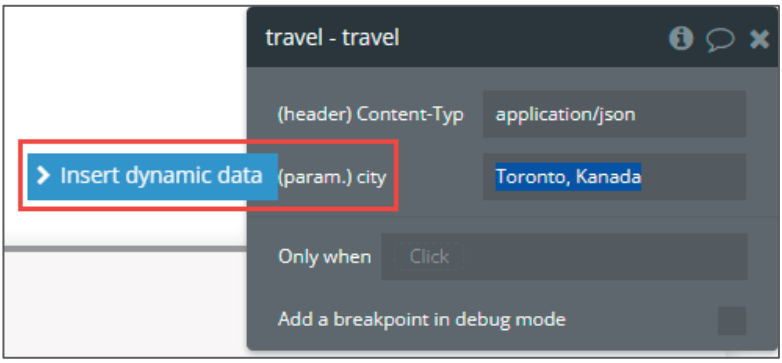
30. Następnie ponownie kliknij przycisk *Click here to add an action*, przewiń listę w dół do pozycji *Element Actions* i wybierz opcję *Set State*. Z listy rozwijanej *Element* wybierz pozycję *Text A*. Z listy rozwijanej *Custom state* wybierz pozycję *itinerary_details*. W polu *Value* ustaw wartość *Results of step 1* i wybierz pozycję *itinerary*. Dzięki temu wartość pola *Text A* będzie ustawiana na wartość `itinerary` zawartą w obiekcie JSON zwracanym przez wywołanie utworzonej wcześniej funkcji chmurowej.



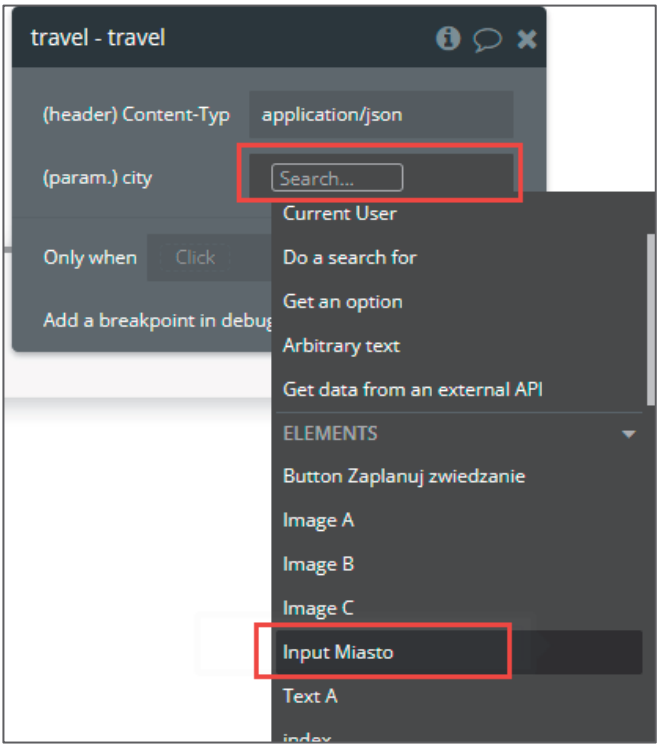
Rysunek 6.21. Konfigurowanie wywołania API w Bubble



Rysunek 6.22. Pomyślny wynik inicjalizacji wywołania API



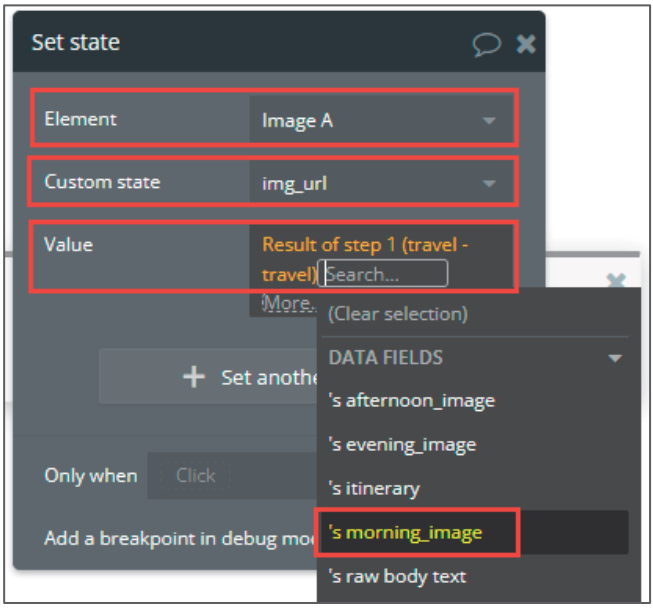
Rysunek 6.23. Dodawanie dynamicznych danych do wywołania API (1)



Rysunek 6.24. Dodawanie dynamicznych danych do wywołania API (2)

31. Następnie ponownie kliknij przycisk *Click here to add an action*, przewiń listę w dół do pozycji *Element Actions* i wybierz opcję *Set State*. Z listy rozwijanej *Element* wybierz pozycję *Image A*. Z listy rozwijanej *Custom state* wybierz pozycję *img_url*. W polu *Value* ustaw wartość *Results of step 1* i wybierz pozycję *morning_image*. Dzięki temu wartość pola *Image A* będzie ustawiana na wartość

`morning_image` zwracaną przez funkcję chmurową i pochodzącą z wcześniejszego wywołania interfejsu API OpenAI Images (rysunek 6.25).

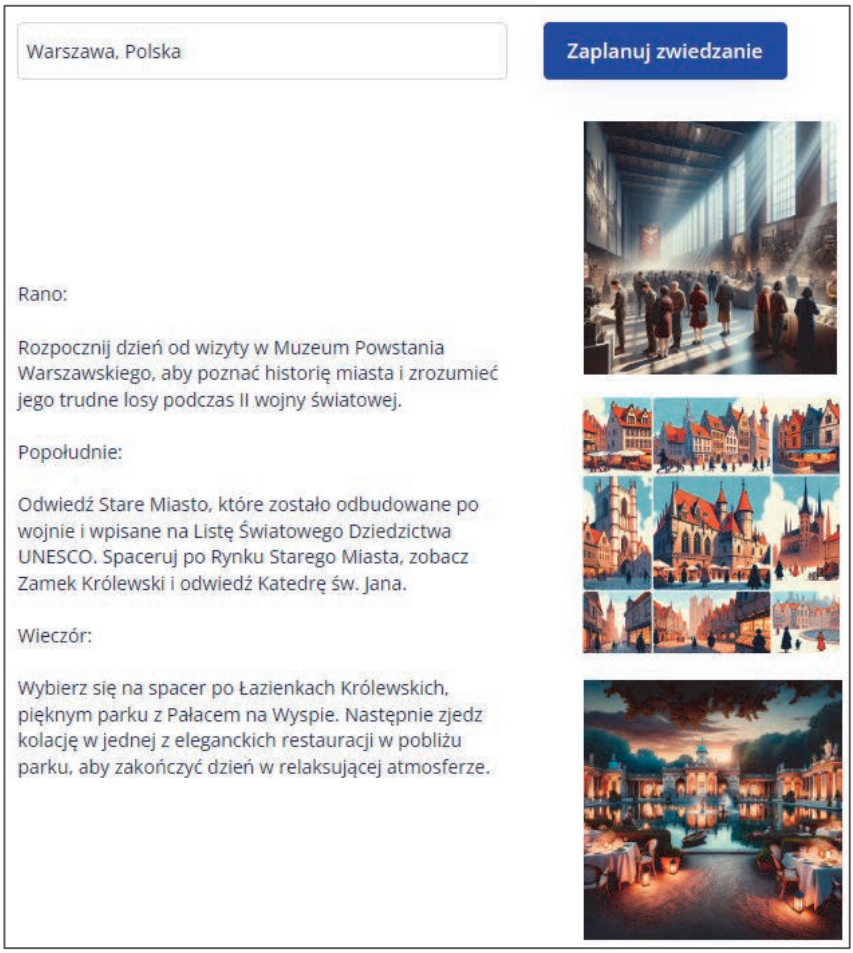


Rysunek 6.25. Przypisywanie niestandardowego stanu w Bubble

32. Potwórz etap 31. dwukrotnie dla elementów *Image B* i *Image C*, przypisując im odpowiednio obrazy *afternoon_image* i *evening_image*.
33. Twoja inteligentna aplikacja Bubble jest gotowa. Aby sprawdzić, czy działa, kliknij przycisk *Preview* znajdujący się w prawym górnym rogu okna; pojawi się nowa strona z Twoją aplikacją.
34. W polu tekstowym *Miasto* wpisz dowolną nazwę miasta (ja wpisałem Warszawa, Polska). Następnie kliknij przycisk *Zaplanuj zwiedzanie*; spowoduje to uruchomienie procesu Bubble i wywołanie funkcji chmurowej.
35. Jeśli wszystko pójdzie dobrze, powinieneś zobaczyć ekran podobny do pokazanego na rysunku 6.26. Przedstawia on plan zwiedzania Warszawy oraz trzy obrazy odpowiadające temu, o czym jest mowa w planie.

Jak to działa?

W tej recepturze utworzyłeś aplikację turystyczną, w której użytkownik może wpisać nazwę dowolnego miasta, aby uzyskać jednodniowy plan zwiedzania wraz z wygenerowanymi przez AI obrazami odpowiadającymi temu planowi. Jak już wspomniano,



Rysunek 6.26. Wyniki aplikacji Bubble

jest to aplikacja *multimodalna*, ponieważ do jej skonstruowania użyłeś zarówno interfejsu OpenAI Chat API, jak i Images API.

Konfigurowanie ustawień aplikacji multimodalnej

Ponieważ jest to aplikacja multimodalna, podczas tworzenia funkcji chmurowej dokonałeś jednej ważnej zmiany — ustawiłeś limit czasu (*Function Timeout*) na 300 sekund (najwyższą dozwoloną wartość; wartością domyślną jest 60 sekund). Ma to kluczowe znaczenie, ponieważ daje aplikacji czas na realizację bardziej złożonych i czasochłonnych zadań. Ze względu na skomplikowaną naturę przetwarzanych danych multimodalnych — łączenie tekstu, obrazów, a może nawet wideo — konieczne jest zapewnienie dodatkowego czasu na parsowanie i analizę danych oraz generowanie odpowiedzi.

Zmiana ta sprawia też, że zaplecze może niezawodnie obsłużyć nagły wzrost ilości lub złożoności danych, co jest typowe w konfiguracjach multimodalnych.

Omówienie kodu związanego z trzema wywołaniami OpenAI API

Warstwa zaplecza zawiera trzy wywołania OpenAI API, które warto omówić bardziej szczegółowo, żebyś wiedział, co dokładnie dzieje się w każdym etapie.

Wywołanie 1

```
response = client.chat.completions.create(
      model="gpt-4o",
      messages=[
    {
      "role": "system",
      "content": "Jesteś pomocnym asystentem, który tworzy jednodniowe
      ↪plany zwiedzania miasta wybranego przez użytkownika. Utwórz tylko
      ↪3 aktywności (rano, po południu, wieczór). Podaj tylko plan
      ↪zwiedzania, nic innego."
    },
    {
      "role": "user",
      "content": "Rzym, Włochy"
    },
    {
      "role": "assistant",
      "content": "Rano: \n\Rozpocznij dzień w Koloseum, jednym
      ↪z najbardziej znanych zabytków Rzymu. Wybierz się na wycieczkę
      ↪z przewodnikiem, aby w pełni docenić jego historię i znaczenie.
      ↪\n\nPopołudnie: \n\nUdaj się do Watykanu. Odwiedź Muzea
      ↪Watykańskie, w których znajduje się ogromna kolekcja dzieł sztuki
      ↪i artefaktów historycznych. Nie przegap Kaplicy Sykstyńskiej,
      ↪słynącej z sufitu z freskiem Michała Anioła.\n\ nWieczór:
      ↪\n\nWybierz się na spokojny spacer do fontanny di Trevi. Pamiętaj,
      ↪aby na szczęście wrzucić do fontanny monetę przez ramię. Następnie
      ↪zakończ dzień pyszną włoską kolacją w jednej z pobliskich
      ↪restauracji."
    },
    {
      "role": "user",
      "content": "Lizbona, Portugalia"
    },
    {
      "role": "assistant",
      "content": "Rano: \n\nRozpocznij dzień wizytą w słynnej wieży Belem,
      ↪wpisanej na Listę Światowego Dziedzictwa UNESCO, skąd roztacza się
      ↪panoramiczny widok na Lizbonę. \n\nPopołudnie: \n\nOdkryj
      ↪historyczną dzielnicę Alfama. Przechadzaj się wąskimi, krętymi
      ↪uliczkami, odwiedź katedrę Se i zjedz tradycyjny portugalski lunch
```

```
        ↪w lokalnej tawernie.\n\nWieczór: \n\nUdaj się na kolację do Bairro
        ↪Alto, artystycznej dzielnicy miasta. Następnie wybierz się na
        ↪pokaz fado - tradycyjnego portugalskiego tańca - w jednym
        ↪z lokalnych barów."
      },
      {
        "role": "user",
        "content": city
      },
    ],
        temperature=0.64,
        max_tokens=1024,
        top_p=1,
        frequency_penalty=0,
        presence_penalty=0
      )

      itinerary = response.choices[0].message.content
```

Jest to standardowe wywołanie czatu GPT-4, takie jak to, które wykonywałeś w poprzednich recepturach, ale zauważ, że do wywołania został dołączony długi *dziennik czatu*. W szczególności podałeś OpenAI dwa przykłady wejścia i wyjścia, aby *dostroić* model — patrz tabela na następnej stronie.

W rozdziale 1., rozdziale 2. i rozdziale 4. dowiedziałeś się, jak można użyć **dziennika czatu** do dostrajania generowanych odpowiedzi, i dokładnie to zrobiłeś w tym przykładzie. Uzyskana odpowiedź jest zapisywana w zmiennej `itinerary`.

Wywołanie 2

W drugim wywołaniu prosisz OpenAI API o utworzenie podpowiedzi do generowania obrazów (DALL-E) na podstawie planu zwiedzania, który został utworzony przez poprzednie wywołanie. Mówiąc ściślej, nakazujesz wygenerować trzy podpowiedzi dla modelu DALL-E (jedna dotycząca poranka, jedna dotycząca popołudnia i jedna dotycząca wieczora) rozdzielone pionową kreską (|):

```
response = client.chat.completions.create(
      model="gpt-4o",
      messages=[
        {
          "role": "system",
          "content": "Jesteś pomocnym asystentem, który tworzy podpowiedzi
          ↪dla DALL-E na podstawie planu zwiedzania. Podpowiedzi powinny
          ↪być krótkie. Utwórz jedną podpowiedź dotyczącą poranka, jedną
          ↪dotyczącą popołudnia i jedną dotyczącą wieczora. Podpowiedzi
          ↪dla DALL-E powinny być rozdzielone znakiem \"|\"."
        },
        {
          "role": "user",
```

Wejście	Wyjście
Rzym, Włochy	**Rano:** **Rozpocznij dzień w Koloseum, jednym z najbardziej znanych zabytków Rzymu. Wybierz się na wycieczkę z przewodnikiem, aby w pełni docenić jego historię i znaczenie.** **Popołudnie:** **Udaj się do Watykanu. Odwiedź Muzea Watykańskie, w których znajduje się ogromna kolekcja dzieł sztuki i artefaktów historycznych. Nie przegap Kaplicy Sykstyńskiej, słynącej z sufitu z freskiem Michała Anioła.** **Wieczór: Wybierz się na spokojny spacer do fontanny di Trevi. Pamiętaj, aby na szczęście wrzucić do fontanny monetę przez ramię. Następnie zakończ dzień pyszną włoską kolacją w jednej z pobliskich restauracji.**
Lizbona, Portugalia	**Rano:** **Rozpocznij dzień wizytą w słynnej wieży Belem, wpisanej na Listę Światowego Dziedzictwa UNESCO, skąd roztacza się panoramiczny widok na Lizbonę.** **Popołudnie:** **Odkryj historyczną dzielnicę Alfama. Przechadzaj się wąskimi, krętymi uliczkami, odwiedź katedrę Se i zjedz tradycyjny portugalski lunch w lokalnej tawernie.** **Wieczór: Udaj się na kolację do Bairro Alto, artystycznej dzielnicy miasta. Następnie wybierz się na pokaz fado - tradycyjnego portugalskiego tańca - w jednym z lokalnych barów.**

```
            "content": itinerary
          }
        ],
        temperature=0.64,
        max_tokens=1024,
        top_p=1,
        frequency_penalty=0,
        presence_penalty=0
    )
    dalle_prompts = response.choices[0].message.content
```

Nawet jeśli model zacznie „halucynować”, użytkownik nigdy nie zobaczy wyników tego wywołania — są one używane tylko do generowania obrazów przez model DALL-E.

Wywołanie 3

```
…
dalle_prompts_list = response.choices[0].message.content.split('|')
```

```
image_urls =  []
for prompt in dalle_prompts_list:
    response = client.images.generate(
            model="dall-e-3",
            prompt=prompt,
            size="1024x1024",
            quality="standard",
            n=1
        )
image_urls.append(response.data[0].url)
```

W ostatnim wywołaniu kolejno bierzesz trzy podpowiedzi dla modelu DALL-E, które zostały wygenerowane w poprzednim wywołaniu, i przekazujesz je do OpenAI Images API. Robisz to w pętli. W Pythonie pętla `for` jest strukturą programistyczną, która pozwala wykonać blok kodu wiele razy, zwykle z pewnymi zmianami w każdej iteracji. W tym kontekście systematycznie przetwarzasz wszystkie podpowiedzi dla modelu DALL-E. W każdej iteracji pętla bierze podpowiedź z listy, wysyła ją do OpenAI Images API, a następnie przechodzi do następnej podpowiedzi, aż przetworzy wszystkie.

Zauważ, że zamiast biblioteki `client.chat` używasz biblioteki `client.images`, ponieważ do generowania obrazów potrzebny jest model DALL-E. Wynik każdego wywołania zapisujesz na liście o nazwie `image_urls`. Na koniec funkcja chmurowa zwraca dane wyjściowe o następującej strukturze JSON:

```
result = {
        'itinerary': itinerary,
        'morning_image': image_urls[0],
        'afternoon_image': image_urls[1],
        'evening_image': image_urls[2]
    }
return result
```

W tej recepturze utworzyłeś inteligentną aplikację, która łączy wiele modeli, wiele wywołań API oraz dostrajanie modelu. Ogólnie rzecz biorąc, w niniejszym rozdziale zbudowałeś dwie użyteczne aplikacje, a zajęło Ci to tylko kilka godzin.

Postęp technologii polega na dostosowaniu jej tak, abyśmy nawet jej nie zauważali i by mogła stać się częścią codziennego życia!

Bill Gates

Firma OpenAI pozostaje niekwestionowanym liderem badań nad sztuczną inteligencją. Dzięki udostępnianym przez nią rozwiązaniom tworzenie innowacyjnych aplikacji, takich jak chatboty, wirtualne asystenty, generatory treści i narzędzia zwiększające produktywność, stało się o wiele prostsze. Bezsprzecznie powszechna dostępność technologii AI zmienia zasady gry!

Receptury zawarte w tym zbiorze ułatwią Ci budowę szerokiej gamy inteligentnych aplikacji. Zaczniesz od podstaw OpenAI API — konfiguracji, uwierzytelniania i kluczowych parametrów — po czym szybko przejdziesz do nauki korzystania z najważniejszych elementów API. Następnie przyjdzie czas na zaawansowane receptury, dzięki którym poprawisz wrażenia użytkownika i dopracujesz dane wyjściowe. Dowiesz się, jak wdrażać aplikacje i przygotować je do publicznego użytku. Nauczysz się również budowania inteligentnych asystentów opartych na specjalistycznej wiedzy, a także aplikacji multimodalnych dostosowanych do Twoich specyficznych potrzeb.

W książce:

- podstawy interfejsu OpenAI API, jego możliwości i ograniczenia
- konfiguracja OpenAI API krok po kroku
- zaawansowane funkcje, w tym komunikat systemowy i dostrajanie
- integracja OpenAI API z istniejącymi aplikacjami i procesami
- projektowanie aplikacji wykorzystujących w pełni możliwości OpenAI API

Henry Habib doradza firmom w zakresie budowy narzędzi korzystających ze sztucznej inteligencji. Jest zapalonym instruktorem i edukatorem internetowym, prowadzi też projekty techniczne w dużych bankach i organizacjach rządowych. Jest zwolennikiem rewolucji no-code i dużych modeli językowych. Mieszka z żoną w Toronto, w wolnym czasie grywa w tenisa.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-1758-3
9 788328 917583
Cena: 67,00 zł
packt