Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka Algorytmiczna (INA)**

PRACA DYPLOMOWA
INŻYNIERSKA

Wykorzystanie asystenta głosowego do komunikacji z serwisem analizującym dane pochodzące z aplikacji do kontroli systemów hydroponicznych

Anna Szutkowska

Opiekun pracy
dr inż. Anna Lauks-Dutka

Słowa kluczowe: Amazon Web Services, asystent głosowy, programowanie bezserwerowe

WROCŁAW 2024

STRESZCZENIE

Celem pracy dyplomowej było zaprojektowanie rozszerzenia asystenta głosowego Alexa. Miało ono za zadanie ułatwić użytkownikowi dostęp do danych udostępnianych przez aplikację do kontroli hodowli hydroponicznej. Dane te obejmują pomiary i stan przełączników. Na zaimplementowany system składają się dwie części - projekt interfejsu oraz kod źródłowy uruchamiany przy każdym wywołaniu umiejętności, przy którym wykorzystano nowoczesne rozwiązania chmurowe. Do napisania aplikacji skorzystano z języka Python w wersji 3.9. Aplikację zaimplementowano przy skorzystaniu z Serverless Framework i platformy AWS Lambda.

ABSTRACT

The purpose of BSc thesis was to design an extension for Alexa voice assistant. It had a task to make it easier for a customer to access data provided by a hydroponics systems controlling app. The data includes measurements and states of relays. Implemented system involved two aspects - interface design and source code ran every time the skill was invoked, where the modern cloud technologies have been used. To implement this application Python in version 3.9 was used. The app has been designed using Serverless Framework and AWS Lambda platform.

SPIS TREŚCI

WPROWADZENIE

Tematem pracy jest aplikacja łącząca system asystenta głosowego Amazon Alexa z systemem do kontroli uprawy. Wykorzystując istniejące technologie ma ona za zadanie ułatwić użytkownikowi dostęp do danych pomiarowych czy zarządzania aparaturą wspomagającą utrzymywanie odpowiedniego stanu hodowli.

Projekt hodowli, o który mowa, nosi nazwę Luna i jest rozwijany przez Koło Naukowe Robotyków KoNaR na Politechnice Wrocławskiej. Polega on na stworzeniu alternatywy dla tradycyjnej uprawy roślin wykorzystując hydroponikę. Jest to nowoczesna metoda hodowli, która nie wykorzystuje gleby. Zastosowane rozwiązanie polega na umieszczeniu roślin w wełnie mineralnej i dostarczaniu im wody z rozpuszczonymi składnikami odżywczymi. Pozwala to na szybszy wzrost roślin i mniejsze zużycie wody niż tradycyjne rozwiązania. Dzięki temu, że uprawa jest kontrolowana, rośliny otrzymują dokładnie taką ilość związków mineralnych, jaką potrzebują, przez co w pełni wykorzystują swój potencjał genetyczny[3].

Dotychczasowe nadzorowanie odbywało się poprzez wejście na aplikację webową, zalogowanie się i włączenie odpowiedniej zakładki na stronie. Może to sprawiać problem osobom, które nie mają dostępu do komputera czy telefonu. Stąd narodził się pomysł użycia asystenta głosowego Alexa – systemu głośnomówiącego, który znaleźć się może w każdym telefonie bądź urządzeniu Echo Dot. Do obsługi takiego urządzenia niewymagane jest użycie rąk czy monitora, nie ma również potrzeby przechodzenia przez klasyczny proces logowania. Wystarczy zwrócić się do niego za pomocą mowy przy użyciu słów-kluczy.

Alexa jest to korzystająca z rozwiązań chmurowych usługa firmy Amazon[6]. Pozwala ona używać komend głosowych do obsługi głośnika za pośrednictwem urządzeń z asystentem Amazon Alexa. Alexa może informować nas o aktualnej pogodzie, godzinie, sprawdzać naszą dostępność w kalendarzu czy współpracować z innymi urządzeniami by stworzyć tzw. inteligentny dom.

Wykorzystując przetwarzanie języka naturalnego (ang. *natural language processing, NLP*) i konwertując mowę na łatwe do zinterpretowania przez komputer symbole Alexa rozpoznaje, kiedy zostaje wywołana i o co zostanie poproszona. Nagranie naszej mowy zostaje przesłane do serwerów Amazon, gdzie zostaje rozbite na poszczególne słowa i wyłoniony zostaje sens wypowiedzi. Na podstawie użytych wyrazów uruchomiona zostaje odpowiednia aplikacja. Następnie serwery Amazon wysyłają odpowiedź zwrotną z wiadomością, która zostaje odczytana przez urządzenie[6].

Amazon umożliwia napisanie własnej aplikacji i zdefiniowanie, w jaki sposób ma zostać ona wywołana. Poszerzona w ten sposób zostaje pula aplikacji, do których ma dostęp

urządzenie. Taka aplikacja nosi nazwę umiejętność (ang. *skill*) i pozwala na napisanie dowolnego rozszerzenia, które znaleźć może zastosowanie w prywatnych jak i komercyjnych celach. Ograniczeniem jest tu jedynie nasza wyobraźnia. Aktualnie przez Alexę obsługiwanych aplikacji jest ponad sto trzydzieści tysięcy[6].

Amazon udostępnia deweloperom obszerną bazę informacji na temat umiejętności, dokładną dokumentację oraz konsolę, ułatwiającą jej projektowanie. Korzystając z tego portalu zdefiniować należy interfejs głosowy naszej umiejętności (ang. *voice user interface*), które określa nam, jakimi słowami wywoływana jest nasza aplikacja, oraz jakie polecenia może ona obsłużyć. Zdefiniowanie interfejsu to jednak nie wszystko. Należy bowiem napisać logikę i obsługę wymyślonej umiejętności[1]. Można to zrobić na kilka sposobów (m.in. postawić własny serwer), jednak na potrzeby pracy wykorzystano w tym celu funkcję Lambda przy użyciu frameworka Serverless oraz platformy AWS Lambda. Wady i zalety zastosowanego rozwiązania przedstawiono w rozdziale 1.

Ostatnim krokiem jest połączenie funkcji z aplikacją do kontroli systemu hodowli hydroponicznej. Taka komunikacja odbywa się za pomocą RestAPI z autoryzacją opartą o mechanizm JSON Web Token. Wymiana informacji obejmuje spis dostępnych czujników, przełączników oraz kontrolę nad częścią z nich.

Praca składa się z trzech rozdziałów. Rozdział pierwszy obejmuje analizę i opis składowych wykorzystanych do budowy niniejszego systemu – zasadę komunikacji pomiędzy Alexą a chmurą Amazon oraz chmurą Amazon a zaprojektowaną umiejętnością. Opisana jest tam również sama zasada działania umiejętności, współpraca interfejsu z funkcją oraz połączenie jej z aplikacją do kontroli systemu hydroponicznego. Kolejny rozdział skupia się na implementacji. Przedstawiono tam diagramy klas, użyte technologie oraz zdefiniowane wywołania. Ostatni rozdział obejmie prezentację przykładów użycia, sposób wdrożenia systemu oraz zaimplementowane metody zautomatyzowania budowania aplikacji.

1. ANALIZA SYSTEMU

W tym rozdziale przedstawiono zasadę działania systemu Alexa oraz sposób, w jaki następuje komunikacja z projektowaną aplikacją. Opisano również kwestię bezpieczeństwa umiejętności oraz funkcji Lambda.

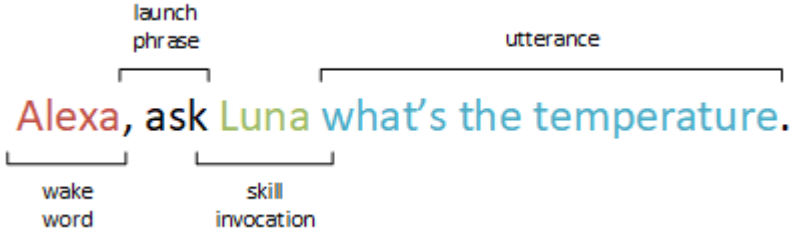
1.1. CEL I ZAŁOŻENIA PROJEKTU

Na cel niniejszej pracy składają się dwa główne zadania. Pierwsze z nich polega na zdefiniowaniu wywołań i zaimplementowaniu podstawowej aplikacji, która uruchamiana będzie przez wcześniej zaprojektowaną umiejętność systemu Alexa. Drugim zadaniem jest połączenie tejże aplikacji z systemem kontroli czujników hodowli hydroponicznej. Została ona napisana przy wykorzystaniu platformy AWS Lambda oraz frameworka Serverless.

1.2. DZIAŁANIE SYSTEMU AMAZON ALEXA

Alexa jest jednym z najpopularniejszych asystentów głosowych. Przeprowadzona przez Insider Intelligence ankieta pokazała, że Amazon Alexa jest najczęściej używanym asystentem głosowym w większości badanych grup wiekowych, ustępując wśród młodszych użytkowników jedynie Siri od Apple[8]. Przewaga Alexy jednak leży w dostępności. Siri jest to asystent głosowy dedykowany dla urządzeń z systemem operacyjnym iOS. Z kolei Amazon Alexa dostępna jest na urządzeniach Echo Dot, ale również na każdym telefonie z zainstalowaną aplikacją Amazon Alexa.

Alexa rozpoczyna swoje działanie w momencie, kiedy użytkownik wybudzi ją za pomocą odpowiednio ułożonych względem siebie słów, których przykładowy porządek zaprezentowano na rysunku 1.1. Urządzenie Alexa nagrywa nasze wypowiedzi, by następnie wysłać je do serwerów Amazona, gdzie zanim zostaną przetłumaczone na zrozumiałe dla



Rysunek 1.1: Przykładowe wywołanie umiejętności Luna

komputera symbole, przechodzą przez kilka kluczowych etapów. Pierwszym z nich jest przetwarzanie sygnałów, które przyczynia się do oczyszczenia nagrań z zakłóceń, takich jak na przykład dźwięk telewizora. Kolejnym zadaniem jest wykrycie słowa wybudzającego nasze urządzenie do działania (ang. wake word). Jest ono niezbędne do włączenia Alexy, bo dopiero, gdy takie słowo zostanie wykryte, analizuje się pozostałą część wypowiedzi. Następnie jest ona wysyłana do Amazon Cloud, gdzie konwertowana jest z wersji audio na tekst. Jako że Alexa jest urządzeniem spersonalizowanym, rozróżniającym zwracających się do niego użytkowników, analizowana jest na tym etapie tonacja i częstotliwość głosu osoby korzystającej z Alexy. Drugim w kolejności słowem analizowanym przez Alexę jest inwokacja (ang. *invocation*). Jest to słowo klucz, które uruchamia odpowiednią umiejętność. Jest to niezbędna część, definiowana na początku tworzenia rozszerzenia. Ostatnią częścią zdania, wymaganą do poprawnego użycia umiejętności, jest tzw. sformułowanie (ang. *utterance*). To za jego pomocą Alexa rozpoznaje zamiar użytkownika i na jego podstawie formułuje odpowiedź[5].

Po przeanalizowaniu wypowiedzi Alexa łączy się z naszą umiejętnością za pomocą mechanizmu żądanie/odpowiedź (ang. request/response) po interfejsie HHTPS. Kiedy użytkownik wywoła umiejętność, otrzymuje ona żądanie POST z niezbędnymi parametrami będącymi treścią elementu ciała (ang. body). To na podstawie tych wartości generowana jest przez aplikację odpowiedź[1]. Proces ten opisano dokładniej w rozdziale 1.6.

1.3. INTERFEJS I SERWIS UMIEJĘTNOŚCI

Każda umiejętność składa się z dwóch komponentów - interfejsu oraz serwisu. Interfejs umiejętności opisuje, w jaki sposób umiejętność ma zostać wywołana. Do zaimplementowania tej części użyto portalu dewelopera Alexa. Jest to narzędzie udostępniane przez firmę Amazon na stronie internetowej `http://developer.amazon.com`. Korzystając z konsoli zarządzania po zalogowaniu uzyskano dostęp do konfigurowania umiejętności. Znajdują się tam opcje zmiany nazwy umiejętności, jej rodzaj czy podstawowy język, który będzie ją obsługiwał.

Najważniejszą częścią interfejsu jest budowa modelu. Sprowadza się ona do zdefiniowania intencji (ang. *intents*), czyli czynności, które mają zostać obsłużone przez umiejętność i które należy zaimplementować. Należy również uzupełnić informacje o zbiorze wypowiedzi obsługiwanych przez umiejętność oraz o wykorzystywanych przez nie danych. Domyślne intencje, które zdefiniowane są po stworzeniu umiejętności obejmują:

- Cancel,
- Help,
- Stop,
- NavigateHome,
- Fallback.

Kluczową częścią każdej umiejętności poza interfejsem jest tzw. serwis umiejętności. To on odpowiada za procedurę obsługi wcześniej zdefiniowanych intencji. Jest on wywoływany za każdym razem, kiedy Alexa rozpozna, że użytkownik użył jednej z dostępnych wypowiedzi i ma za zadanie znaleźć na nią odpowiedź. Do zbudowania serwisu skorzystano z platformy AWS Lambda i Serverless Framework.

1.4. WYKORZYSTANIE PLATFORMY AWS LAMBDA

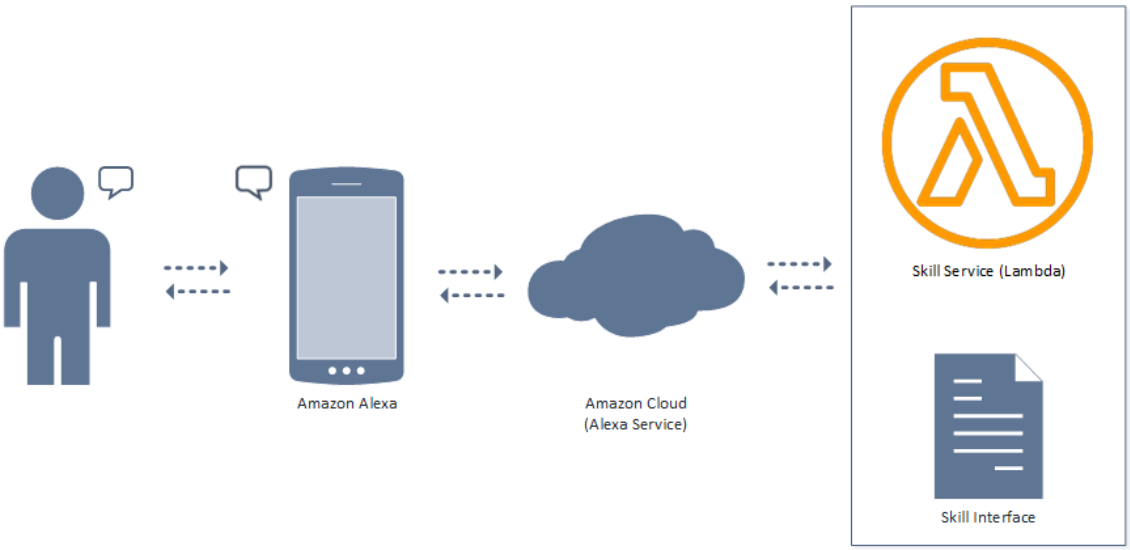
Amazon Webservices Lambda (AWS Lambda) jest dostarczaną przez Amazon platformą obliczeniową. Służy ona do zarządzania i tworzenia aplikacji bez konieczności konfigurowania serwerów i klastrów. Opiera się na architekturze sterowanej zdarzeniami (ang. *event-driven*), które odbywają się w Amazon Web Services i wywołują odpowiednie funkcje. Takie rozwiązanie nosi nazwę Funkcja jako usługa, w skrócie FaaS (ang. *Function-as-a-Service*) i polega na wykonaniu napisanego przez użytkownika kodu w chmurze. To dostawca usługi (ang. *Cloud Service Provider*), udostępnia i zarządza środowiskiem obliczeniowym - infrastrukturą, oprogramowaniem, systemem operacyjnym.

Rozwiązanie FaaS jest w przypadku projektowanej aplikacji bardzo korzystne. Po pierwsze koszty utrzymania są bardzo niskie. Koszty generują pojedyncze żądania - wywołania zdarzeń. W praktyce oznacza to, że płaci się za wykorzystane zasoby, a nie, jak w przypadku rozwiązań serwerowych, również za czas pomiędzy aktywnymi okresami. Kolejną zaletą wykorzystania funkcji jest ich skalowalność. Dostawca usługi umożliwia dopasowanie liczby uruchamianych kopii funkcji w przypadku zwiększonego zapotrzebowania. Ostatnią, najbardziej kluczową dla deweloperów, korzyścią wynikającą z wykorzystania tej technologii jest modułowość kodu. Tworzona aplikacja może wywoływać, w zależności od potrzeb, jedną lub nawet kilkaset współpracujących ze sobą funkcji. Pozwala to na podzielenie kodu na poszczególne funkcjonalności.

AWS Lambda obsługuje języki Java, Go, PowerShell, Node.js, C#, Python i Ruby, jednak udostępnione jest również API, które pozwala na napisanie kodu funkcji w większości pozostałych języków programowania. Ponad 50% funkcji na platformie napisanych zostało w języku Python[2].

1.5. FRAMEWORK SERVERLESS

Framework jest to platforma ułatwiająca budowę aplikacji, udostępniając programiście szkielet struktury i mechanizmu jej działania. W projekcie wykorzystano framework Serverless, czyli open-source'owe narzędzie ułatwiające zarządzanie FaaSami, czyli opisanymi rozdziale 1.4 rozwiązaniami chmurowymi, jak na przykład AWS Lambda, Azure Functions czy Google Cloud Functions. Framework ten dostarcza interfejs CLI (ang. *Command Line Interface*), czyli wiersz poleceń, za pomocą którego można zbudować, usunąć czy



Rysunek 1.2: Schemat komunikacji pomiędzy użytkownikiem a aplikacją

w dowolny inny sposób zarządzać funkcją. Interfejs jego wiersza poleceń oparty jest na Node.js, który wymagany jest do korzystania z narzędzia.

1.6. KOMUNIKACJA POMIĘDZY ALEXĄ A LAMBDĄ

Schemat komunikacji pomiędzy wszystkimi częściami systemu pokazano na rysunku 1.2. By umożliwić przekazywanie informacji pomiędzy Alexą a funkcją AWS Lambda należy korzystając z konsoli dewelopera zdefiniować endpoint (z ang. punkt końcowy). W architekturze REST jest to urządzenie, które posiada możliwość łączenia się z siecią celem odwoływania się do niego. Amazon udostępnia dwa rodzaje punktów końcowych, służących do komunikacji - HTTPS lub AWS Lambda ARN. Niezależnie od wybranego rodzaju punktu końcowego, komunikacja odbywa się po interfejsie HTTPS. Wybranie typu „AWS Lambda ARN" pozwala natomiast skorzystać z usług chmurowych firmy Amazon. Opcja „HTTPS" wymaga od nas skonfigurowania własnej usługi Web Service. Na potrzeby poniższej pracy wybrano drugą opcję. ARN, czyli Amazon Resource Name, to unikatowy identyfikator zasobu w Amazon Web Services. W tym przypadku jest to identyfikator funkcji AWS Lambda, która zostaje wywołana, kiedy użytkownik wejdzie w interakcję z naszą umiejętnością.

Identyfikator ARN przyjmuje następujący format:

`arn:partition:service:region:account-id:resource-type:resource-id`.

Poszczególne części identyfikatora oddzielone są dwukropkiem i oznaczają:

- grupę regionu AWS,
- nazwę serwisu,
- kod regionu AWS,

- numer konta AWS,
- typ i identyfikator zasobu.

Korzystając ze zdefiniowanego punktu końcowego, Alexa wysyła żądanie POST z parametrami zawartymi w wypowiedzi użytkownika w formie obiektu JSON. Najważniejszą częścią wiadomości jest treść ciała (ang. body). Zawsze składa ona z wersji, kontekstu (opisuje stan urządzenia, z którego wygenerowano zapytanie) oraz żądania. To ostatnie zawiera kluczowe informacje do zbudowania serwisu umiejętności, bo to tam określona jest wywołana intencja.

Listing 1.1: Przykładowy obiekt typu JSON otrzymywany przez funkcję podczas wywołania umiejętności

```
{
  "version": "1.0",
  "session": {

  },
  "context": {

  },
  "request": {
    "type": "LaunchRequest",
    "requestId": "amzn1.echo-api.request.requestID",
    "locale": "en-GB",
    "timestamp": "YYYY-MM-DDTHH:MM:SSZ",
    "shouldLinkResultBeReturned": false
  }
}
```

Pole `request.type` określa typ żądania:

- `LaunchRequest` - użytkownik wydał umiejętności polecenie nie korzystając z żadnej konkretnej intencji,
- `CanFulfillIntentRequest` - zapytanie, czy umiejętność potrafi obsłużyć intencję z brakującymi danymi,
- `IntentRequest` - użytkownik skorzystał z jednej ze zdefiniowanych intencji,
- `SessionEndedRequest` - poinformowanie umiejętności o zakończeniu sesji.

1.7. KOMUNIKACJA Z APLIKACJĄ DO KONTROLI UPRAWY

Komunikacja pomiędzy serwisem a aplikacją do kontroli uprawy odbywa się po interfejsie HTTP (ang. *Hypertext Transfer Protocol*). Odpowiada on za wymianę informacji pomiędzy komunikującymi się urządzeniami. Serwer HTTP używa portu TCP 80 do pola żądania. W pracy wykorzystano dwa rodzaje żądań: GET oraz POST. Metoda GET stosowana jest w przypadku pobrania zasobów, natomiast metoda POST służy do wysyłania informacji od klienta do serwera.

Do zweryfikowania zapytań wydawanych przez serwis do aplikacji kontrolującej pomiary zastosowano JSON Web Token. Jest to ustandaryzowany sposób przesyłania informacji pomiędzy serwerem a klientem. Składa się on z trzech części:

- nagłówka - zawiera metadane opisujące typ tokenu,
- ładunku (ang. *payload*) - zawiera dane do przesłania drugiej stronie, np. dane logowania niezbędne to autoryzacji dostępu użytkownika,
- sygnatury - zwanej również podpisem, zawiera ona dane niezbędne do zweryfikowania autentyczności tokenu.

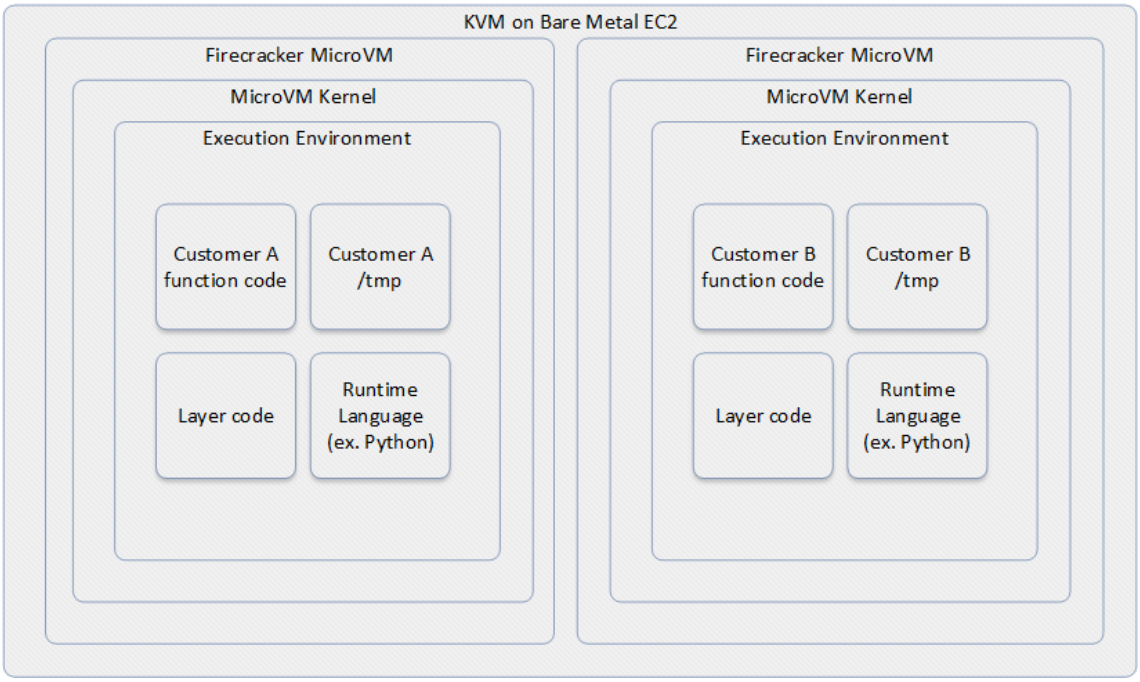
1.8. BEZPIECZEŃSTWO

Korzystając z umiejętności opublikowanych na stronie Amazon należy zwrócić uwagę na kwestię bezpieczeństwa. Zapewnia się, że przy wywołaniu umiejętności, ale bez skorzystania ze zdefiniowanych intencji, nasze zapytanie nie zostanie przekazane do serwisu umiejętności. Istnieje jednak dużo obszarów, w których istnieją luki w zabezpieczeniach, umożliwiające pobór danych wrażliwych czy poważne nadużycia.

Każda umiejętność przed opublikowaniem przejść musi przez proces weryfikujący spełnienie określonych wymagań. Należą do nich między innymi:

- zgodność z wytycznymi dotyczącymi prywatności proponowanych przez Alexę (znajduje się tam na przykład zakaz korzystania z nazw firmowych bez przedstawienia dowodu afiliacji),
- zaliczenie wszystkich wymaganych testów zarówno od strony użytkownika jak i od strony interfejsu, czy
- spełnienie wszystkich wytycznych dotyczących bezpieczeństwa udostępnianych na zewnętrznych serwerach usług (czyli na przykład w przypadku, kiedy nie korzysta się z platformy AWS Lambda).

Przy każdorazowej zmianie konfiguracji umiejętności należy ponownie przejść cały proces weryfikujący. Nie dotyczy to jednak zmian wprowadzonych w kodzie źródłowym aplikacji, co prowadzić może do poważnych nadużyć i niekontrolowanych pogwałceń polityk prywatności[9].



Rysunek 1.3: Schemat środowiska wykonawczego funkcji platformy AWS Lambda

Ważnym aspektem jest również bezpieczeństwo kodu i połączeń pomiędzy użytkownikiem a aplikacją na platformie AWS Lambda. Korzystając z platformy nie ma się wpływu na infrastrukturę, a wszelkie kwestie prywatności i zabezpieczeń pozostawia się w rękach firmy Amazon. Zapewniają oni jednak, że kod aplikacji jest szyfrowany, a poszczególne wywołania funkcji uruchamiane są na izolowanych środowiskach, każde z własnymi zasobami i systemem plików, co zaprezentowano na schemacie 1.3. Środowisko wykonawcze przypisywane jest do poszczególnych kont i uruchamiane jest na maszynie wirtualnej bazującej na jądrze. Nie jest ono współdzielone przez inne funkcje czy użytkowników[11].

2. PROJEKT I IMPLEMENTACJA SYSTEMU

W tym rozdziale przedstawiono szczegółowy projekt systemu, korzystając między innymi z notacji UML (zaprezentowane w pracy diagramy i schematy znajdują się w załączniku A). Opisano obsługę zdarzeń i intencji wywoływanych przez użytkownika korzystającego z asystenta głosowego. Do napisania kodu źródłowego użyto języka Python w wersji 3.9 oraz biblioteki Serverless, która umożliwia konfigurowanie funkcji na platformę AWS Lambda.

2.1. WYMAGANIA

2.1.1. Wymagania niefunkcjonalne

Głównym celem systemu jest osiągnięcie łatwego dostępu do danych przez dowolnego użytkownika. Zastosowanie systemu głośnomówiącego Alexa pozwala na korzystanie z aplikacji osobom z niepełnosprawnościami związanymi z funkcjami ręki czy osobom niedowidzącym. System ma być zrozumiały oraz w razie konieczności dostarczać potrzebne instrukcje dotyczące swojego działania.

Liczba błędów pojawiających się na panelu kontrolnym wynikających z niemożności obsługi intencji powinna nie przekraczać 10% liczby wszystkich wywołań. Jeśli chodzi o solidność systemu, ma on być odporny na wszelkie awarie i utratę danych. Zdefiniowane intencje mają umożliwiać użytkownikowi otrzymanie informacji o wszystkich wykonywanych pomiarach oraz dostępnych modułach.

2.1.2. Wymagania funkcjonalne

Użytkownik powinien mieć dostęp do instrukcji obsługi umiejętności. Powinien również mieć możliwość zapytania o wszystkie dostępne wartości parametrów mierzonych na uprawie hydroponicznej. W razie wystąpienia błędu użytkownik powinien dostać dokładną informację o tym, gdzie on nastąpił - w systemie, czy w sposobie wywołania.

2.2. PLIK KONFIGURACYJNY

Plik konfiguracyjny aplikacji jest najważniejszą jego częścią. Zdefiniowane są tu kluczowe dla funkcji elementy, takie jak identyfikator umiejętności, z którą ma się połączyć, czy nazwa funkcji wywoływanej przy każdym uruchomieniu. W przypadku tego projektu

skorzystano z jednej funkcji o nazwie Thesis, której główna metoda znajduje się w pliku `handler.py` i nosi nazwę `lambda_handler`. Identyfikator umiejętności pobrano z pliku zmiennych środowiskowych. Aplikację zaprojektowano z myślą o środowisku z językiem Python w wersji 3.9 dla regionu Europa Zachodnia i platformy AWS. Wywoływana jest ona za pomocą zdarzeń typu AlexaSkill.

Listing 2.1: Plik konfiguracyjny aplikacji napisanej przy pomocy Serverless Framework

```
org: ankaszutek
app: thesis
service: aws-python-lambda
useDotenv: true

frameworkVersion: '3'

provider:
  name: aws
  runtime: python3.9
  region: eu-west-1

functions:
  thesis:
    handler: handler.lambda_handler
    events:
      - alexaSkill:
          appId: ${env:ALEXA_SKILL_ID}
          enabled: true
    environment:
      LUNA_APP_URL: ${env:LUNA_APP_URL}
      LUNA_APP_USERNAME: ${env:LUNA_APP_USERNAME}
      LUNA_APP_PASSWORD: ${env:LUNA_APP_PASSWORD}

plugins:
  - serverless-python-requirements

custom:
  pythonRequirements:
    dockerizePip: non-linux
```

Kod pliku konfiguracyjnego zaprezentowano na listingu 2.1. Pole `org` oznacza nazwę organizacji, odpowiadającej za wybudowanie aplikacji. Najczęściej przybiera ona formę

nazwy użytkownika, do którego należy aplikacja. Kolejne pole app określa nazwę aplikacji. Ważnym elementem jest również pole `provider.runtime`, które definiuje na jakim środowisku zbudowany jest program.

Dla wywołania aplikacji kluczowym jest pole `functions`. To dam określamy liczbę, nazwy i nazwy metod obsługujących zdarzenia funkcji. Zmienne zewnętrzne niezbędne do korzystania z funkcji określone są w polu `functions.environment` i pobierane są w trakcie budowania aplikacji z pliku ze zmiennymi środowiskowymi.

2.3. WYWOŁANIE FUNKCJI AWS LAMBDA

Równie ważną do zaimplementowania częścią każdej funkcji platformy AWS Lambda jest handler, czyli metoda, od której aplikacja zaczyna swoje wykonywanie, której fragment zaprezentowano na listingu 2.2. Jej nazwa określona jest w pliku `serverless.yml`, pliku konfiguracyjnym niezbędnym do zbudowania funkcji, opisanym w rozdziale 2.2.

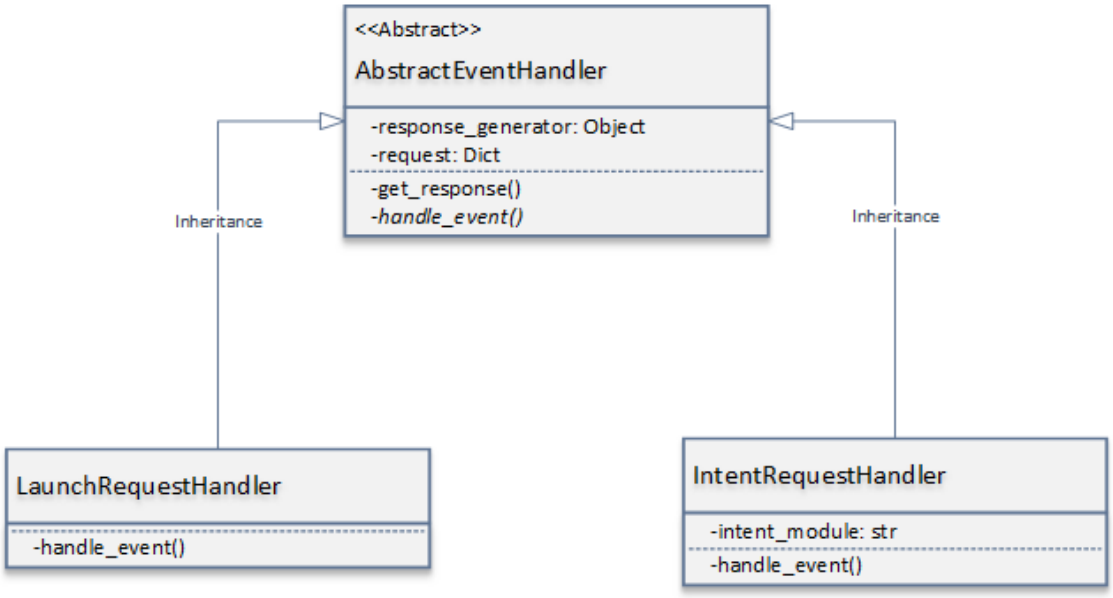
Listing 2.2: Główna metoda-handler, wywoływana przy każdorazowym uruchomieniu funkcji AWS Lambda

```
1  def lambda_handler(event: Dict[str, Any], context: str) -> str:
2      request_type = event["request"]["type"]
3      logging.Logger(context)
4      if request_type == "LaunchRequest":
5          handler = LaunchRequestHandler(event["request"])
6      elif request_type == "IntentRequest":
7          handler = IntentRequestHandler(event["request"])
...
13     return handler.get_response()
```

Decyduje ona, jakie zdarzenie zostało wywołane i oddelegowuje jego obsługę do odpowiedniej klasy. Ostatnim jej zadaniem jest uzyskanie wygenerowanej odpowiedzi zwrotnej.

2.4. ANALIZA WYWOŁANEGO ZDARZENIA

W rozdziale 1.6 przedstawiono przykładowy obiekt typu JSON, który wysyła Alexa przy każdorazowym wywołaniu umiejętności. W zależności od wypowiedzi użytkownika, określające typ żądania pole `request.type` przyjmuje różne wartości. Najważniejszym rodzajem żądania jest `IntentRequest`. Oznacza ono, że użytkownik zwrócił się do asystenta głosowego z intencją, która pokrywa się z jedną z wcześniej zdefiniowanych. Kolejnym znaczącym rodzajem żądania jest `LaunchRequest`, który używany jest przy każdorazowym rozpoczęciu sesji. Za obsługę tych i pozostałych rodzajów żądań odpowiada klasa abstrak-



Rysunek 2.1: Diagram klas odpowiadających za obsługę podstawowych zdarzeń

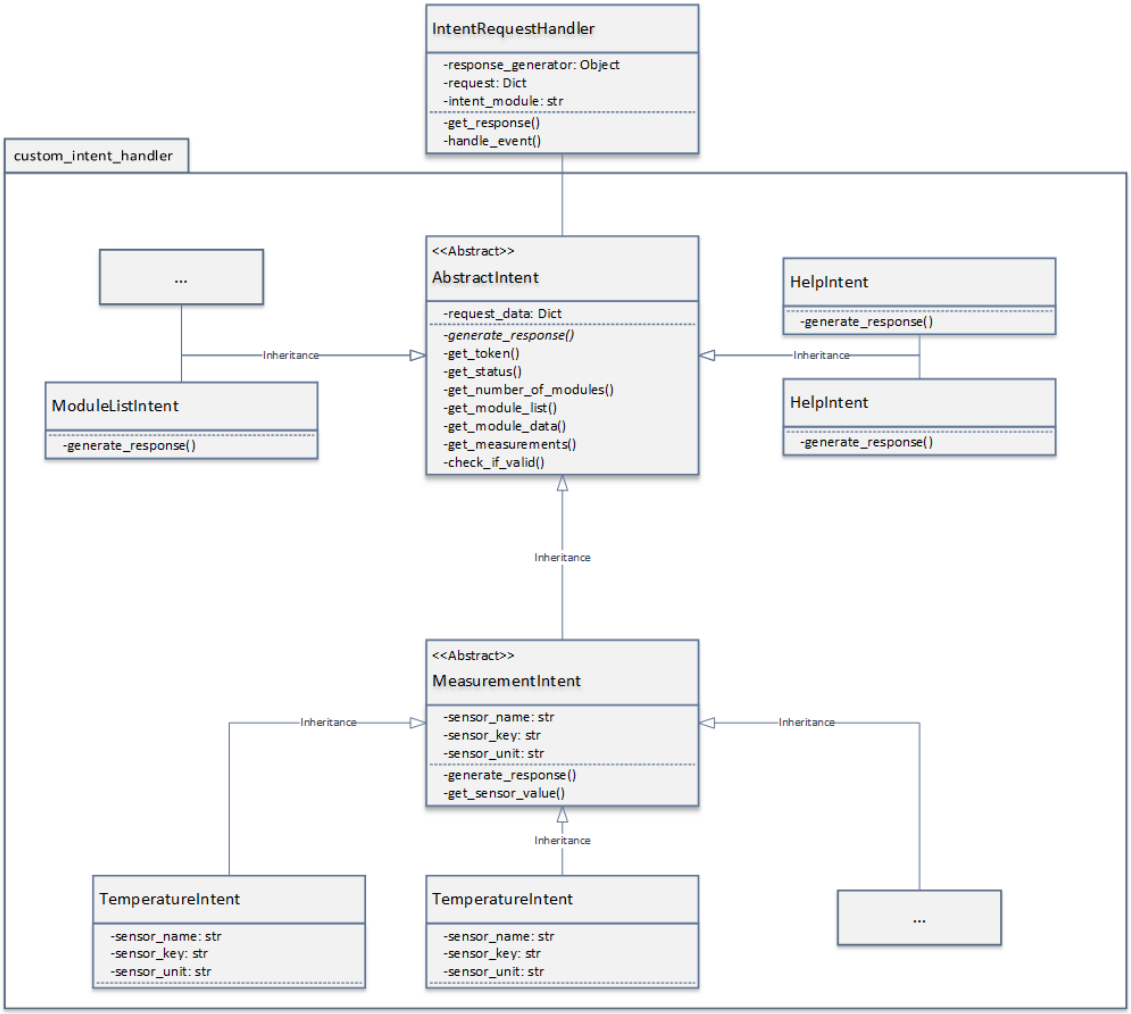
cyjna o nazwie `AbstractEventHandler`, zaprezentowana na diagramie 2.1. Posiada ona dwie metody - `handle_event`, odpowiada za obsługę zdarzenia i generowanie odpowiedzi oraz `get_response`, która tę odpowiedź udostępnia dalej do Alexy.

Klasa `IntentRequestHandler` odpowiada za obsługę intencji. Jest ona kluczową częścią, decydującą o tym, która metoda ma zostać wywołana. Kod źródłowy 2.3 przedstawia sposób, w jaki przetwarzane jest żądanie i jak wyciągana jest informacja o zastosowanej przez użytkownika intencji.

Listing 2.3: Metoda klasy IntentRequestHandler obsługująca żądanie typu IntentRequest

```
def _handle_event(self) -> None:
    module = importlib.import_module(self.intent_module)
    intent_name = self.request["intent"]["name"]
    try:
        intent = class_(self.request["intent"].get("slots"))
        self.response_generator.generate_response(
            intent.generate_response()
        )
    except (UserOperationError, EndpointError) as e:
        self.response_generator = ErrorResponse()
        self.response_generator.generate_response(str(e))
```

W zależności od wartości pola `self.request["intent"]["name"]` tworzony jest obiekt odpowiedniej klasy, jednej z kilku dziedziczących po klasie `AbastractIntent`, znajdujących się w pakiecie określonym zmienną `self.intent_module`. Pakiet ten nosi nazwę `custom_intent_handler` i zawiera wszystkie klasy obsługujące intencje, jak poka-



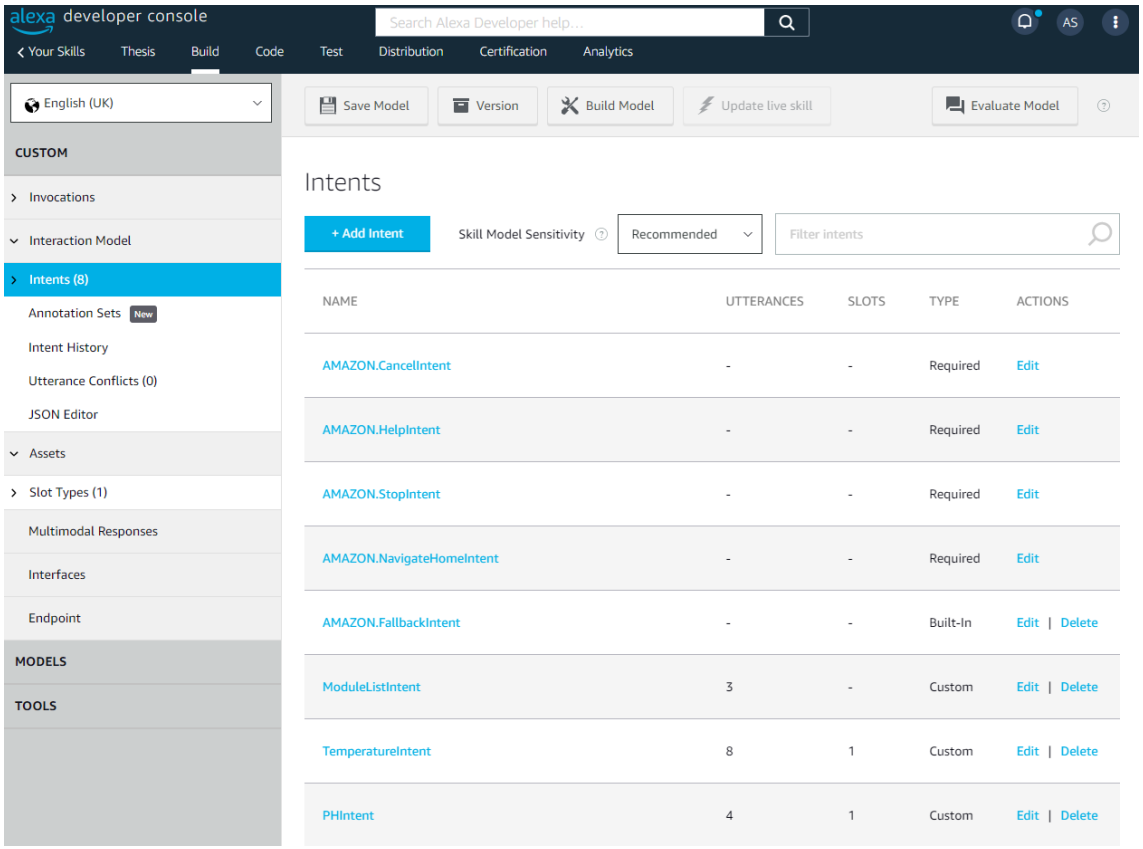
Rysunek 2.2: Diagram klas odpowiadających za obsługę intencji

zano na diagramie 2.2. Informację o wszelkich danych przekazanych wraz z wypowiedzią przekazano jako argument przy tworzeniu nowego obiektu. Na koniec wygenerowano odpowiedź zwrotną do Alexy.

2.5. INTENCJE

Listę dostępnych w umiejętności intencji zdefiniowano korzystając z interfejsu umiejętności, czyli przy użyciu platformy deweloperskiej udostępnianej przez Amazon. Przy tworzeniu nowego elementu skorzystać można z dostępnych wzorów (np. AMAZON.HelpIntent) lub napisać go w całości od nowa (np. ModuleListIntent), jak zaprezentowano na rysunku 2.3.

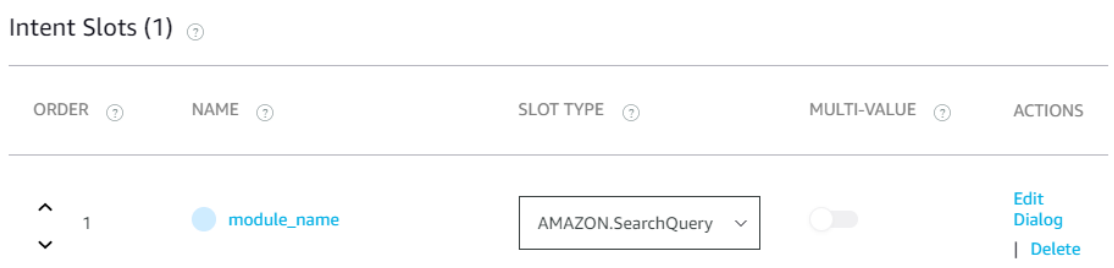
Każda intencja składa się ze sformułowań (ang. *utterance*). Jest to lista wypowiedzi, która powoduje wywołanie intencji. W przypadku użycia złych słów i braku zgodności wypowiedzi użytkownika z założeniami, żadne żądanie nie jest wykonywane.



Rysunek 2.3: Widok konsoli dewelopera Alexy na listę zdefiniowanych intencji

Niektóre z intencji mogę posiadać dodatkowe pola zawierające informacje niezbędne do wypełnienia przez użytkownika. Definiuje się je podczas tworzenia intencji - określa się nazwę zmiennej, w której dana wartość będzie przechowywana, oraz jej typ. Na rysunku 2.4 pokazano przykładowe zastosowanie tych pól w przypadku TemperatureIntent - prośby o podanie temperatury wody w danym module.

Przykładowy diagram sekwencyjny, pokazujący w jaki sposób następuje wywołanie umiejętności oraz połączenie między poszczególnymi częściami systemu, pokazano na rysunku 2.5. Zauważyć tam można, że w wypowiedzi użytkownika zabrakło informacji o nazwie modułu, w którym ma zostać sprawdzona temperatura. Za wykrycie tego błędu odpowiada interfejs umiejętności. Odpowiada on żądaniem o wypełnienie brakujących da-



Rysunek 2.4: Przykładowe pole na dodatkowe dane intencji TemperatureIntent

nych. Po otrzymaniu wszystkich niezbędnych informacji wywoływana jest funkcja Lambda i przesyłane jest do niej odpowiednie zapytanie. Następnie po stronie aplikacji leży analiza oraz wygenerowanie odpowiedzi zwrotnej do użytkownika.

Tak jak opisano to na diagramie 2.2, klasa odpowiadająca za TemperatureIntent - `TemperatureIntentHandler`, wraz z kilkoma innymi dziedziczy po abstrakcyjnej klasie `MeasurementIntent`. Klasa ta odpowiada za pobranie szczegółowych informacji na temat konkretnego pomiaru - temperatury, pH czy liczby całkowitych rozpuszczonych substancji stałych (TDS). Poszczególne klasy dziedziczące posiadają jedynie dodatkowe informacje na temat jednostek czy nazwy parametru.

Listing 2.4: Fragment kodu klasy AbstractMeasurementIntent

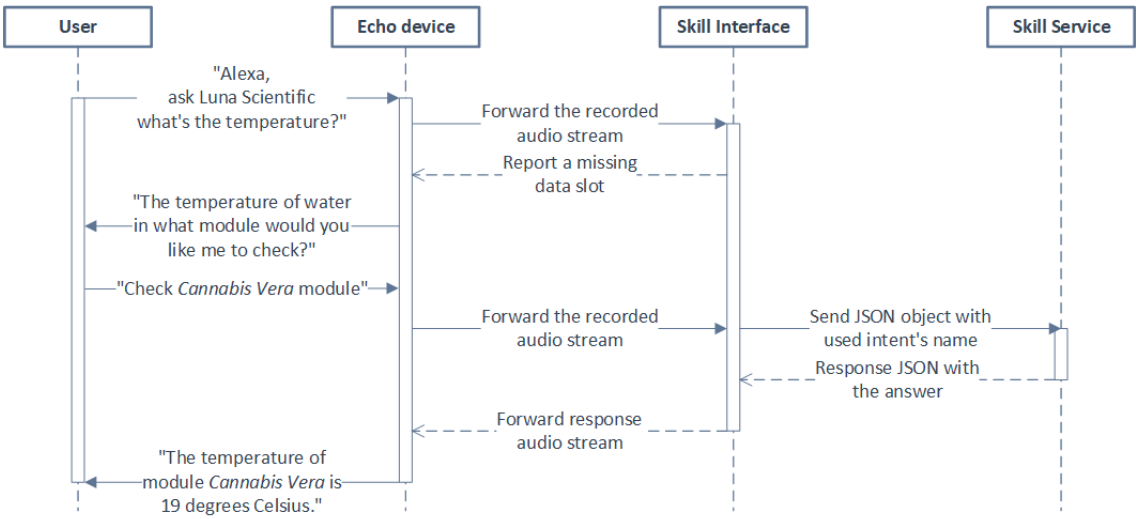
```
class AbstractMeasurementIntent(AbstractIntent):
    sensor_name: str = None
    sensor_key: str = None
    sensor_unit: str = None

    def generate_response(self) -> str:
        module_name, module_index = self._get_module_data()
        sensor_value = self._get_sensor_value(module_index)
        return f"{self.sensor_name} in module {module_name} is
        ↪ {sensor_value} {self.sensor_unit}"

    def _get_sensor_value(self, module_index: int) -> int:
        value = self._get_measurements(module_index, self.sensor_key)
        if value:
            return value
        raise EndpointError(MEASUREMENT_EMPTY)
```

2.6. ODLEGŁOŚĆ LEVENSHTEINA

Aby zminimalizować liczbę błędów wynikających z niepoprawnej konwersji słów na tekst przez serwis Amazon, zastosowano dodatkowy system znajdujący możliwe dopasowania. W przypadku intencji, które wymagają podania nazwy modułu, obliczono odległości Levenshteina podanego wyrazu i wszystkich istniejących w bazie nazw modułów.

Odległość Levenshteina (znana również jako odległość edycyjna) jest miarą odmienności napisów, która jest stosowana w przetwarzaniu informacji. Określa ona minimalną liczbę działań prostych (wstawienia, usunięcia bądź podmiany pojedynczych liter), które wymagane są by zmienić jedno słowo w drugie[10].

Sposób liczenia miary zaprezentowano w wyrażeniu 2.1. *Tail* oznacza ciąg znaków następujących po pierwszym znaku wyrazu, a `a[0]` oznacza pierwszy znak w wyrazie `a`.



Rysunek 2.5: Diagram sekwencyjny przykładowego zapytania o temperaturę wody danego modułu

Do zaimplementowania metody zastosowano programowanie dynamiczne. Napisany kod przedstawiono na rysunku 2.5. Aby określić, czy dane słowo jest wynikiem błędu konwersji, użyto stałej o wartości 3 określającej liczbę działań prostych wymaganych do otrzymania poprawnego ciągu znaku.

$$lev(a,b) = \begin{cases} |a|, & \text{dla } |b| = 0, \\ |b|, & \text{dla } |a| = 0, \\ lev(tail(a), tail(b)), & \text{dla } a[0] = b[0], \\ 1 + min \begin{cases} lev(tail(a), b), \\ lev(a, tail(b)), \\ lev(tail(a), tail(b)), \end{cases} & \text{w p.p..} \end{cases} \tag{2.1}$$

Listing 2.5: Fragment kodu klasy AbstractIntent odpowiedzialnej za obliczanie odległości Levenshteina

```
@staticmethod
def _levenshtein_distance(module_name: str, match: str) -> int:
  @cache
  def _minimal_distance(first: int, second: int) -> int:
      if first == len(module_name) or second == len(match):
          return len(module_name) - first + len(match) - second
      if module_name[first] == match[second]:
          return _minimal_distance(first + 1, second + 1)
      return 1 + min(
          _minimal_distance(first, second + 1),
```

```
            _minimal_distance(first + 1, second),
            _minimal_distance(first + 1, second + 1)
        )
    return _minimal_distance(0, 0)
```

Istnieje podobna metoda miary nosząca nazwę odległość Damerau-Levenstheina. Do działań prostych opisanych powyżej dodaje ona jedną nową operację - podmianę znaku na sąsiedni. Z badań przeprowadzonych przez Fredericka J. Damerau wynika, że dzięki temu można zmniejszyć liczbę błędów nawet o 80%[4]. W przypadku opisywanego projektu nie miałoby to jednak zastosowania, jako że tego typu błędy wynikają z niepoprawnego zapisania wyrazów na klawiaturze przez człowieka. System przetwarzania mowy zastosowany w asystencie głosowym nie popełniałby tego typu błędów, a jedynie związane z niepoprawnym zrozumieniem zamiaru i wypowiedzi użytkownika.

2.7. POŁĄCZENIE Z APLIKACJĄ LUNY

Za komunikację za pomocą protokołu HTTPS z aplikacją do kontroli uprawy hydroponicznej odpowiada klasa `ApiHandler`(fragment 2.6), skorzystano również z pakietu `requests`, biblioteki w języku Python obsługującej zapytania HTTP. Wszelkie dane potrzebne do autoryzacji użytkownika oraz adres URL aplikacji pobierane są z pliku `.env`. Przy każdorazowym stworzeniu instancji klasy `ApiHandler` generowany jest token dostępu JWT. Wykorzystywany jest on we wszelkich żądaniach celem weryfikacji dostępu. Odbywa się to poprzez wysłanie żądania GET, zawierającego dane niezbędne do zalogowania - nazwę oraz hasło użytkownika. W odpowiedzi otrzymano token autoryzacyjny, wykorzystywany dalej w nagłówku `Authorization` wysyłanych zapytań.

Listing 2.6: Fragment kodu klasy ApiHandler

```
class ApiHandler:
    url = os.environ.get("LUNA_APP_URL")
    body = {"username": os.environ.get("LUNA_APP_USERNAME"),
            "password": os.environ.get("LUNA_APP_PASSWORD")}

    def __init__(self):
        self._generate_token()

    def _generate_token(self) -> None:
        try:
            authorization = self._authorize()
        except HTTPError:
            raise EndpointError
        if authorization.status_code != 204:
```

```
            self.token = authorization.json()["access"]
            self.headers = {"Authorization": f"Bearer {self.token}"}
        else:
            raise HTTPError

    def _authorize(self) -> Response:
        response = requests.post(f"{self.url}api-auth/token/",
        ↪  data=self.body)
        response.raise_for_status()
        return response
```

W podobny sposób wysyłane są pozostałe żądania. Zaprezentowana na fragmencie kodu 2.7 metoda odpowiada za pobranie wybranych rekordów pomiarów. Przedział pomiarów może być określony przez przedział czasowy podany w trakcie wywoływania intencji. Za pomocą metody GET pobierane są dane o module o konkretnym identyfikatorze, definiowanym na podstawie podanej przez użytkownika nazwy modułu.

Listing 2.7: Fragment kodu klasy ApiHandler

```
def get_module_history(self, module_id: int,
                start_date: str = None, end_date: str = None,
                count: str = None) -> list:
    body = self.body | {"start_date": start_date, "end_date": end_date,
    ↪  "measurements_count": count}
    response =
    ↪  requests.get(f"{self.url}hydroponics/measurements/{module_id}/",
                        headers=self.headers, data=body)
    response.raise_for_status()
    return response.json()
```

W przypadku problemu z komunikacją lub otrzymaną odpowiedzią wywoływany jest błąd `EndpointError`. Generowana jest odpowiednia informacja, która dostarcza niezbędnych danych pozwalających zweryfikować, na jakim etapie komunikacji nastąpił problem.

3. INTEGRACJA I WDROŻENIE

Rozdział ten opisuje w jaki sposób zaprezentowane wcześniej aplikacje zostały zainstalowane oraz w jaki sposób należy z nich korzystać. Opisano w nim również metody testowania oraz automatyzacji budowania funkcji AWS Lambda. Przedstawiono rysunki, na których widać dane wyjściowe przykładowych wywołań umiejętności czy konsolę do kontroli i konfiguracji aplikacji Serverless.

3.1. WDROŻENIE I INSTALACJA

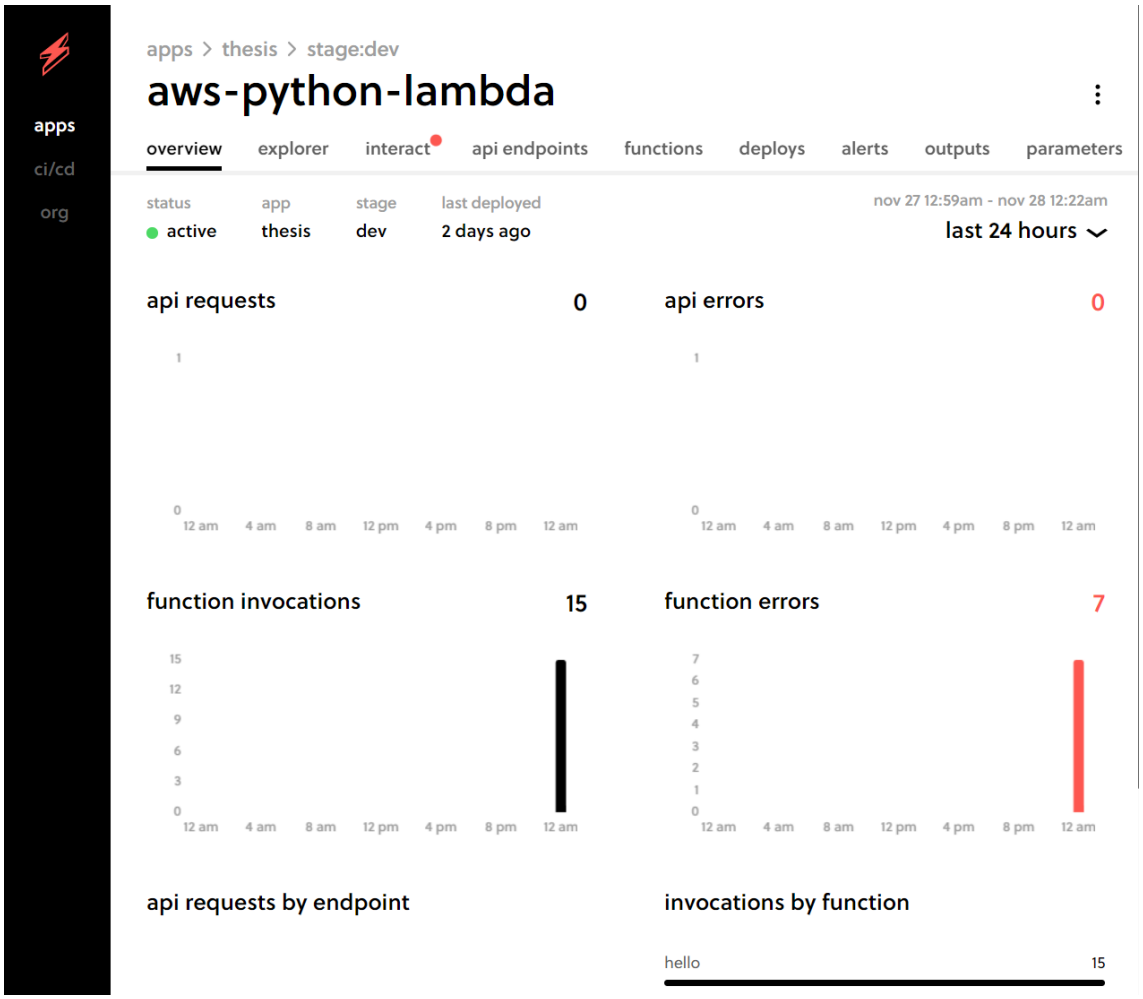
3.1.1. Aplikacja Serverless

Do zainstalowania Serverless skorzystano z NPM, menadżera plików środowiska Node.js, za pomocą komendy `npm install -g serverless`. Niezbędnym krokiem po zainstalowaniu narzędzia jest udostępnienie uprawnień do konta AWS, z którego skorzystano przy budowaniu umiejętności. Po skonfigurowaniu dostępu i podaniu odpowiednich haseł oraz kluczy przejść można do pisania aplikacji.

Po napisaniu kodu należy zbudować zaimplementowaną aplikację. Odbywa się to poprzez wykonanie komendy `serverless deploy`, której wynik zaprezentowano na listingu 3.1. Na konsoli otrzymano ukazany poniżej wynik. Korzystając z opcji `--verbose` uzyskać można bardziej szczegółowe informacje dotyczące budowania, m.in. nazwy poszczególnych etapów czy identyfikator ARN potrzebny do skonfigurowania punktów końcowych umiejętności. Aby móc korzystać z najnowszej wersji aplikacji, należy powtórzyć etap budowania jej za każdym razem, kiedy wprowadzane są zmiany do kodu źródłowego.

Listing 3.1: Dane wyjściowe po wybudowaniu aplikacji za pomocą komendy Serverless

```
$ serverless deploy

Deploying thesis to stage dev (eu-west-1)

Service deployed to stack thesis-dev (36s)

functions:
  luna_scientific: thesis-dev-luna_scientific (47 kB)
```



Rysunek 3.1: Widok ze strony app.serverless.com do kontroli aplikacji

3.1.2. Budowanie umiejętności

Aby móc przetestować działanie umiejętności należy ją najpierw zbudować. Po zdefiniowaniu punktów końcowych, intencji oraz wybudowaniu aplikacji, czas na interfejs. Interfejs umiejętności budowany jest przy pomocy konsoli dewelopera Alexa korzystając z opcji „Build". Budowanie to trwa kilka sekund i należy je ponowić za każdym razem, kiedy dokonano zmiany w interfejsie.

3.1.3. Statystyki

Ważnym etapem kontroli podczas wdrażania aplikacji jest kontrolowanie statystyk i przesyłanych danych. Odbywa się to z poziomu strony Serverless Framework oraz kosoli dewelopera Alexa. To pierwsze szczególnie przydatne umożliwia nam wgląd do liczby wywołań czy błędów aplikacji, co zaprezentowano na rysunku 3.1. Otrzymano tam również historię konfigurowania aplikacji czy możliwość testowania obsługi własnych żądań.

3.2. PRZYKŁADOWE WYWOŁANIE UMIEJĘTNOŚCI

Konsola dewelopera Alexa umożliwia testowanie wywołania umiejętności za pomocą tekstu. Nie wymaga instalowania dodatkowych pakietów czy publikowania umiejętności, przez co ułatwia proces usprawniania aplikacji na każdym etapie produkcji. Po zdefiniowaniu intencji oraz wdrożeniu aplikacji możemy wybudować model umiejętności. Następnie korzystając z zakładki Test na stronie umiejętności w konsoli dewelopera uzyskujemy dostęp do panelu czatu. Ma on za zadanie symulować wymianę wypowiedzi pomiędzy użytkownikiem a Alexą. Ma się również wgląd do przesyłanych pomiędzy interfejsem a serwisem żądań i zapytań. Widok panelu czatu do testowania przedstawiono na rysunku 3.2.

3.3. TESTOWANIE

W projekcie zadbano o wysokie pokrycie kodu testami jednostkowymi. Do napisania testów w języku Python skorzystano z pakietu `unittest` - udostępnianego w standardowej bibliotece frameworka. Posiada on stałą, wymuszaną na programiście strukturę testów, co zwiększa ich czytelność. Każdy zbiór testów należy przedstawić w postaci klasy dziedziczącej po klasie `unittest.TestCase`, jak zaprezentowano to na fragmencie 3.2. Pojedyncze testy w tym zbiorze należy przedstawiać jako metody tej klasy[7].

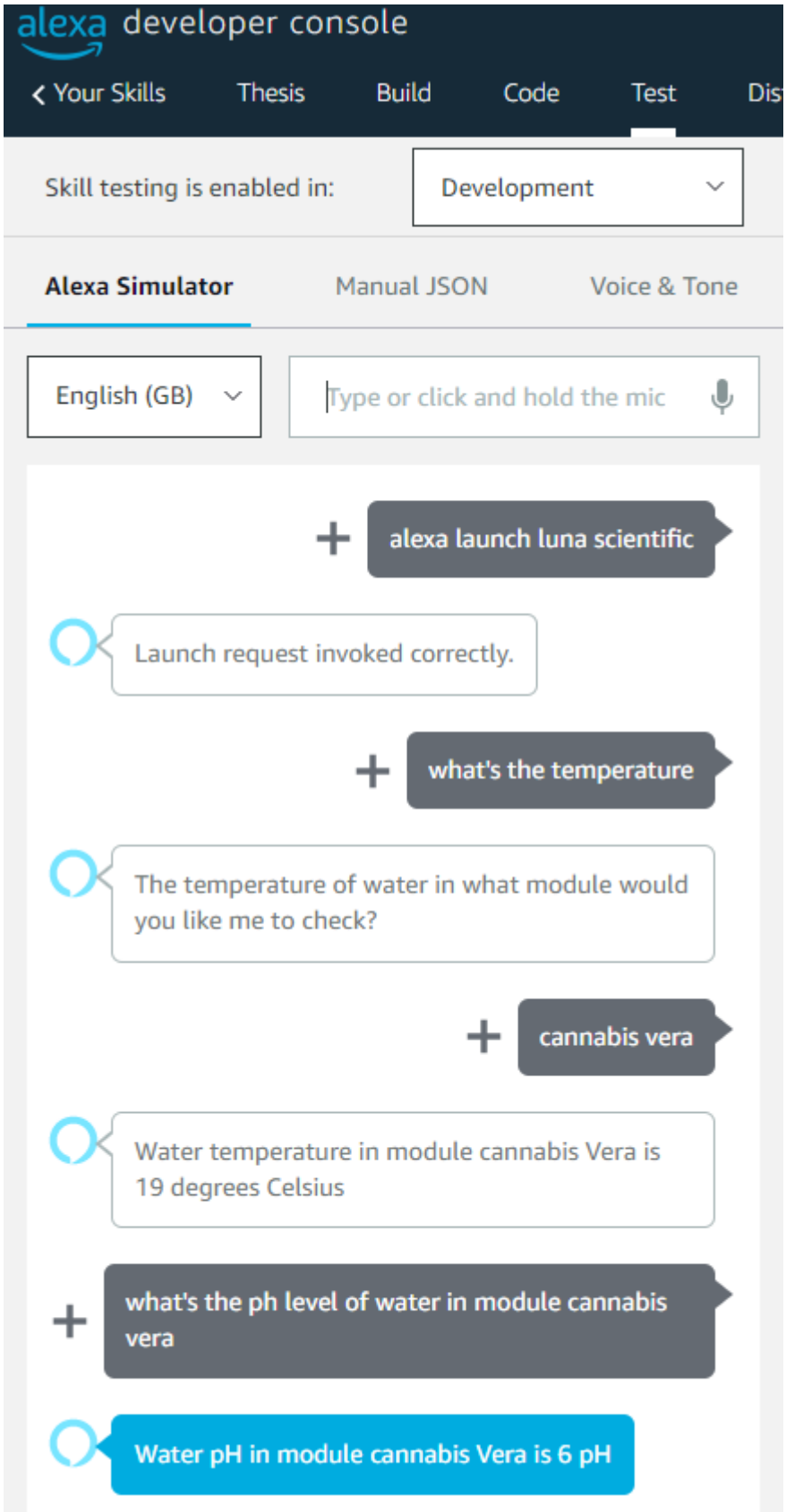
Unittest pozwala na automatyczne wykrywanie, czy dany plik zawiera testy. Kluczem w tym przypadku jest nazwa pliku, w którym testy się znajdują. Aby zostały wykryte przez framework, powinny zaczynać się prefiksem `test`.

Osiągnięto pokrycie kodu o wartości powyżej 90%. Wszystkie testy jednostkowe znajdują się w folderze `/tests`.

Dodatkowe testy, które zaimplementowano, obejmują testy integracyjne, sprawdzające poprawność interakcji pomiędzy poszczególnymi modułami. Testy te wykonano na etapie CI, opisanym w rozdziale 3.4. Zadaniem ich jest przetestowanie reakcji aplikacji na przykładowe żądanie.

Listing 3.2: Fragment kodu testów jednostkowych klasy AbstractEventHandler

```
class SomeEventHandler(AbstractEventHandler):

    def _handle_event(self) -> None:
        self.response_generator.generate_response("Event Handler Test")


class TestAbstractEventHandler(TestCase):

    def setUp(self) -> None:
        self.handler = SomeEventHandler(request)
        self.response = response
```



Rysunek 3.2: Wywołanie umiejętności Luna Scientific z intencją zapytania o temperaturę w danym module

```
12
13      def test_get_response(self) -> None:
14          self.assertEqual(json.dumps(self.response),
            ↪  self.handler.get_response())
```

3.4. CIĄGŁA INTEGRACJA

Ważną częścią projektowania i budowania tego typu aplikacji jest automatyzacja. Proces CI/CD, który zastosowano podczas implementacji tego projektu, polega na ciągłej integracji (ang. *Continuous Integration*) oraz ciągłego wdrażania (ang. *Continuous Delivery*) zmian. Jest to dość nowy temat nabierający popularności, którego liczba zastosowań zwiększa się z roku na rok[12].

Ciągła integracja to proces, w którym pracująca nad projektem grupa deweloperów wprowadza częste zmiany, które są każdorazowo testowane, a cały projekt budowany[12]. W przypadku poniższej pracy czynnikiem, który podlegał temu procesowi, jest serwis umiejętności, czyli aplikacja napisana w języku Python przy użyciu Serverless Framework. Testy obejmują testy jednostkowe i integracyjne, natomiast proces budowania odbywał się zgodnie z opisem w rozdziale 3.1.1.

Do automatyzacji tego procesu skorzystano z usługi GitHub CI, czyli narzędzia udostępnianego przez serwis internetowy GitHub, który główną rolą jest kontroli wersji Git. Główną częścią takiego rozwiązania, jest tzw. rurociąg (ang. *pipeline*), czyli nic innego jak plan kolejno wykonywanych przez środowisko kroków skonfigurowanych w skrypcie.

Skrypt ten aby zostać wykryty przez GitHub CI znajdować się musi w folderze `.github/workflows/`. Określa on sposób wywołania rurociągu. W przypadku tego projektu odbywa się to przy okazji każdego wypchnięcia (ang. *push*) zmian na gałąź `main` do repozytorium. Uruchamiane kroki obejmują między innymi wykonanie testów jednostkowych. W momencie niepomyślnego zakończenia się co najmniej jednego z nich, uzyskujemy negatywną wiadomość w repozytorium. Pozwala to na szybką informację o błędach w kodzie i na dłuższą metę - na zminimalizowanie czasu potrzebnego na analizę i poprawę kodu źródłowego.

Listing 3.3: Plik konfiguracyjny zawierający skrypt uruchamiany przez GitHub CI

```
1  name: CI Pipeline
2
3  on:
4    push:
5      branches: [main]
```

```yaml
jobs:
  ci-pipeline:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v2
      - name: Setup Python
        uses: actions/setup-python@v2
        with:
          python-version: 3.10.0
          architecture: x64
          cache: 'pip'
      - name: Install Dependencies
        run: pip install -r requirements.txt
      - name: Run Tests
        run: python -m unittest
      - name: Check Coverage
        run: pytest --cov=thesis/src tests/
```

PODSUMOWANIE

Temat pracy brzmi „Wykorzystanie asystenta głosowego do komunikacji z serwisem analizującym dane pochodzące z aplikacji do kontroli systemów hydroponicznych”. Głównym jej celem było ułatwienie dostępu do danych pomiarowych hydroponicznej stacji uprawowej projektu Luna. O ile zamierzony cel został osiągnięty, temat nie został jeszcze wyczerpany, a projekt można jak najbardziej dalej rozwijać.

Podczas pracy nad projektem wyłoniono trzy podstawowe problemy:

- zdefiniowanie interfejsu umiejętności,
- zaimplementowanie serwisu umiejętności - bezserwerowej aplikacji,
- połączenie wyżej wspomnianej aplikacji z systemem kontroli uprawy.

Wszystkie te punkty powinny ze sobą współgrać, co udało się osiągnąć. System pomyślnie przeszedł testy i jest gotowy do użytkowania. Udało się zbudować solidną bazę możliwych do wywołania intencji. Każda z nich ma od kilku do kilkunastu możliwych wypowiedzi, co pozwala na swobodne korzystanie z umiejętności.

Na chwilę obecną istnieje możliwość zapytania asystenta o:

- listę dostępnych modułów,
- poziom pH, TDS, EC czy temperatury wody w danym module,
- średnie wartości poziomów pH, temperatury itd. w podanym przedziale czasowym,
- stan danego przełącznika,
- pomoc w korzystaniu z umiejętności.

Przedstawione w rozdziale pierwszym technologie wymagały dużego nakładu pracy związanego z ich poznaniem. Są to nowe narzędzia, które poprawnie zastosowane ułatwiają pracę programisty. Istnieje jednak wiele innych platform, które przyczyniłyby się do lepszego wyniku i bezpieczeństwa.

Jedną z takich technologii jest Marshmallow, czyli biblioteka w języku Python umożliwiająca serializację i walidację obiektów JSON. Zastosowanie jej przyczyniłoby się do większej pewności, że wymieniane pomiędzy uczestnikami informacje są w poprawnym formacie. Kolejną możliwością jest wykorzystanie danych środowiskowych i zabezpieczenie je, kiedy są w użyciu za pomocą klucza AWS Key (na chwilę obecną szyfrowane są jedynie dane w spoczynku).

Cały system można rozszerzyć o liczbę intencji, czy dopisać więcej możliwych wypowiedzi do tych już istniejących. Pozwoliłoby to na lepsze pokrycie możliwości i pozwoliło

na uniknięcie sytuacji, kiedy użytkownik wypowie sformułowanie, w którym brakuje jednego słowa i Alexa nie zaliczy go jako poprawne.

Bezserwerową aplikację można by podzielić na dwie współpracujące ze sobą funkcje. Jedna z nich komunikowałaby się z aplikacją do kontroli uprawy, a druga odpowiadałaby za odbieranie i odsyłanie wiadomości do asystenta głosowego Alexa. Ułatwiłoby to kontrolę i rozszerzalność systemu.

BIBLIOGRAFIA

[1] *What is the Alexa Skills Kit?*, https://developer.amazon.com/en-US/docs/alexa/ask-overviews/what-is-the-alexa-skills-kit.html. Dostęp: 2022-12-02.

[2] *The state of serverless*, https://www.datadoghq.com/state-of-serverless-2021/#8. 2021. Dostęp: 2022-11-15.

[3] *Powstaje „Luna" – nowy zautomatyzowany system do uprawy roślin bez gleby*, https://pwr.edu.pl/uczelnia/aktualnosci/powstaje-luna-nowy-zautomatyzowany-system-do-uprawy-roslin-bez-gleby-12507.html. 2022. Dostęp: 2022-11-13.

[4] Damerau, F.J., *A technique for computer detection and correction of spelling errors*, Communications of the ACM. 1964, tom 7, 3, str. 171–176.

[5] Gonfalonieri, A., *How Amazon Alexa works? Your guide to Natural Language Processing (AI)*, https://towardsdatascience.com/how-amazon-alexa-works-your-guide-to-natural-language-processing-ai-7506004709d3. 2018. Dostęp: 2022-11-10.

[6] Kolczyk, E., *Voice AI – how does it work? (based on Amazon Alexa)*, Women In Tech Summit. 2022.

[7] Krekel, H., *How to use unittest-based tests with pytest*, https://docs.pytest.org/en/7.1.x/how-to/unittest.html. 2015. Dostęp: 2022-12-02.

[8] Lebow, S., *Siri, Alexa, and Google Assistant popularity varies across US age groups*, https://www.insiderintelligence.com/content/siri-alexa-google-assistant-popularity-varies-across-us-age-groups. 2021. Dostęp: 2022-11-10.

[9] Lentzsch, C., *Hey Alexa, is this Skill Safe?: Taking a Closer Look at the Alexa Skill Ecosystem* (2021).

[10] Levenshtein, V., *Binary codes capable of correcting deletions, insertions, and reversals*, Soviet Physics-Doklady. 1966, tom 10, 8.

[11] Mayank Thakkar, Marc Brooker, O.S., *Security Overview of AWS Lambda*, Rap. tech., Amazon Web Services. 2022.

[12] Mojtaba Shahin, M.A.B., *Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices*, IEEE Access. 2017.

SPIS RYSUNKÓW

SPIS LISTINGÓW

Dodatki

A. PŁYTA CD

Do pracy dyplomowej załączono płytę CD zawierającą kody źródłowe, diagramy, zrzuty ekranu oraz instrukcję pomagającą wybudować umiejętność na własnym systemie. Poniżej przedstawiono nazwy poszczególnych folderów oraz opisy ich zawartości.

Na płycie znajdują się 3 foldery:

- diagramy - pliki w formacie PNG wszystkich zawartych w pracy diagramów oraz schematów,
- zrzuty ekranu - zawarte w pracy zrzuty ekranu z konsoli dewelopera czy strony aplikacji biblioteki Serverless,
- aplikacja - kod źródłowy aplikacji serwisu umiejętności.

Dodatkowo znajdują się tam również skompresowany folder zawierający kod interfejsu umiejętności oraz plik README.txt, w którym znaleźć można instrukcję. Zawiera ona opis kolejnych kroków niezbędnych do odtworzenia projektu na własnym systemie. Obejmuje to stworzenie umiejętności oraz wybudowanie aplikacji.