Dokumentacja do aplikacji ChatBot CI
Aplikacja obsługuje zapytania programistów o status / wynik integrowania ich zmiany w kodzie
Marcin Krajewski

Aplikacja udostępnia webowy interfejs pod adresem http://ci-bot.pl

Komunikacja w aplikacji:
[zapytanie] : front-end (web) -> chatbot PHP -> MySQLi+baza danych
[odpowiedź] : front-end (web) <- chatbot PHP <- MySQLi+baza danych

Front-end (web):
Responsywna strona wykonana w HTML i css3 z użyciem frameworka Bootstrap v4 umożliwia bezproblemową obsługę aplikacji również na urządzeniach mobilnych.
Pliki odpowiedzialne:
- **index.php** z dołączoną biblioteką Bootstrap v4
- dodatkowe **style.css**.

ChatBot w PHP:
Odpowiedzialne za działanie ChatBota są 2 pliki:
- test_program.php (główny plik programu, podrzędny w stosunku do **index.php**, przez który jest wywoływany) Udostępnia formularz do zapytań. Pełni funkcję analizy zapytań tekstowych z formularza i odpowiada za komunikację z bazą danych. Realizuje też podstawową logikę decyzyjną o zapytania numeryczne.

- inc_if_status_logi_bledy.php – plik odpowiedzialny za logikę decyzyjną programu i dodawanie odpowiedzi ChatBota do bazy danych z historią rozmowy. Plik podłączony do nadrzędnego względem niego **test_program.php** jako rozwinięcie logiki decyzyjnej zapytań numerycznych.

Tab. Pliki które są wynikiem testów jednostkowych modułów użytych później w programie i ich kolejność i występowania w kodzie

Plik testowy modułu	Odpowiednik kodu w aplikacji (oznaczony komentarzem)
test_connect_db.php	//connect();
test_pokaz_historie_rozmow.php	//pokaz_historie_rozmow();
test_no_ci.php	//czy_pyta_numer();
test_status_ci.php	//czy_pyta_status();
test_error_ci.php	//czy_pyta_bledy();
test_logi_ci.php	//czy_pyta_logi();
test_accept_question.php	//czy_accept_question();
test_dodaj_rozmowe_do_db.php	//dodaj_rozmowe_do_db();
test_jaki_status_numeru.php	//jaki_status_numeru();

* Zestawienie nie zawiera pliku testowego **inc_if_status_logi_bledy.php** a w zasadzie jego postaci przed rozwinięciem go o funkcjonalności zapisu odpowiedzi do bazy. Testy oczywiście zostały przeprowadzone nawiązanie do nich będzie w posumowaniu.

Debugowanie i kontrola logiczności odpowiedzi realizowana jest poprzez za komentowaną linijkę:
//echo "</br>$numer, $no_q, $status_q, $error_q, $logi_q, $accept_q</br>"; //wynik analizy pytania

Rozpoznawanie zapytań / stringów od użytkownika
Analizą zapytań tekstowych od użytkownika zajmują się bloki kodu które w wyniku tej analizy ustawiają stany zmiennych wykorzystywanych w późniejszym procesie decyzyjnym.

Tab. Bloki kodów wraz z opisem działania i możliwymi stanami

Nazwa Bloku	Krótki opis działania kodu	Ustawiane stany zmiennych
//czy_pyta_numer();	Sprawdza czy w stringu pytania od użytkownika wystąpił ciąg numeryczny, a pierwszy znaleziony zapisuje jako wartość zmiennej $numer	True: $no_q = 1 False: $no_q = 0
//czy_pyta_status();	Sprawdza czy w stringu pytania od użytkownika wystąpiło szukane słowo kluczowe: status (rozpoznaje słowo kluczowe niezależnie od wielkości liter)	True: $status_q = 1 False: $status_q = 0
//czy_pyta_bledy();	Sprawdza czy w stringu pytania od użytkownika wystąpiły szukane słowa kluczowe: błęd i błąd (rozpoznaje słowa kluczowe niezależnie od wielkości liter)	True: $error_q = 1 False: $error_q = 0
//czy_pyta_logi();	Sprawdza czy w stringu pytania od użytkownika wystąpiło szukane słowo kluczowe: logi (rozpoznaje słowo kluczowe niezależnie od wielkości liter)	True: $logi_q = 1 False: $logi_q = 0
//czy_accept_question();	Sprawdza czy w stringu pytania od użytkownika wystąpiły szukane słowa kluczowe: tak, poproszę, yes, accept (rozpoznaje słowa kluczowe niezależnie od wielkości liter)	True: $accept_q = 1 False: $accept_q = 0

Bloki kodu odpowiedzialne za komunikację z bazą danych
//pokaz_historie_rozmow(); - blok kodu który wyświetla zawartość kolumny **talk** z tabeli **talk_history**. Sprawdza czy istnieje historia, czy zapytanie SQL zostało wykonane poprawnie, na koniec pobiera i wyświetla dane.

//dodaj_rozmowe_do_db(); - blok kodu który występuje wielokrotnie w całym programie ale generalnie w 2 wariantach. Pierwszy występujący tylko raz odpowiedzialny za dodawanie zapytań $_POST['question']; od użytkownika do kolumny **talk** tabeli **talk_history** (ze sprawdzaniem poprawności dodania). Drugi występuje wielokrotnie, za każdym razem jak ChatBot wyświetla odpowiedź string jest dodawany do tej samej kolumny jako następny rekord (ze sprawdzaniem poprawności dodania).

//jaki_status_numeru(); - blok kodu który szuka znaleziony w stringu $_numer zmiany / integracji w tabeli **ci_table** (ze sprawdzaniem poprawności zapytania) , jeśli nie ma wyników to odpowiada że taka zmiana nie istnieje i ustawia wartość zmiennej na pytania o numer na **$no_q = 0** zatrzymując dalsza analizę. Zaś jeśli istnieje taki numer w bazie i są wyniki z zapytania to nadaje im odpowiednie wartości, później używane do wyświetlania danych w odpowiedziach:
$id_integration = $numer_status['id_integration'];
$status_integration = $numer_status['status_integration'];
$error_integration = $numer_status['error_integration'];
$log_integration = $numer_status['log_integration'];

Mechanizm przydzielania odpowiedzi - w uproszczeniu
Elementy w pliku test_program.php

- if ($no_q == 1) { jeśli w stringu wystąpiło pytanie o numer, program sprawdza
    - //jaki_status_numeru(); **patrz opis wyżej**, przyjmując ze jest taka zmiana, pobiera jej dane i załącza plik
        - include('**inc_if_status_logi_bledy.php**'); który decyduje jakie dane wyświetlić
- else { w stringu nie ma numerów i program prosi o podanie poprawnego zapytania o zmianę. Podaje przykład

Elementy w pliku inc_if_status_logi_bledy.php
Zasada przydzielania odpowiedzi: od najszczegółowszych warunków

- //pytanie o numer i status i logi i błędy
  if(($no_q == 1) && ($status_q == 1) && ($logi_q == 1) && ($error_q == 1)){ jeśli występują wszystkie słowa kluczowe w stringu pytania i wcześniej program sprawdził poprawność numeru to podaje odpowiednie dane ale przed tym jeszcze modeluje odpowiedź o błędy zależnie czy
  - if ($error_integration == 0){ to zamiast wartości podaje string: „niema błędów" i zapisuje odpowiedź w bazie danych blokiem //dodaj_rozmowe_do_db();
  - else { podaje normalną wartość błędu string = $error_integration. " i również zapisuje odpowiedź w bazie danych blokiem //dodaj_rozmowe_do_db();

- program ponawia operację dla każdej kombinacji szukanych słów kluczowych zawężając kryterium

- //pytanie o numer i status i logi
  else if (…) {
- //pytanie o numer i status i błędy
  else if (…) {
- //pytanie o numer i logi i błędy
  else if (…) {
- //pytanie o numer i status
  else if (…) {
- //pytanie o numer i logi
  else if (…) {
- //pytanie o numer i błędy
  else if (…) { ten blok kodu podaje poprawne szukane dane: błędy dla szukanej zmiany.
  Ale dla sprawdzenia warunków zadania ChatBot zadaje dodatkowe pytanie: „Czy chcesz zobaczyć logi ?"
  - if($accept_q == 1){ Ten blok jako jedyny korzysta ze zmiennej **$accept_q**. W założeniu odpowiedź „tak" powinna spowodować uzyskanie logów ale że odpowiedź „tak" nie spełnia warunków podstawowych (nie zawiera informacji o numerze zmiany itd.) ChatBot zaproponuje: „Podaj numer zmiany, która Cię interesuje. Posiadam informacje o 7 zmianach. Najlepiej do numeru dopisz co chcesz wiedzieć o statusie, błędach lub logach. Przykład: Chciałbym poznać aktualny status oraz logi ze zmiany numer 3."
- //pytanie o sam numer
  else if (…) { ten blok domyślnie poda poprawny status szukanej zmiany.

- // jak nie padnie ani numer ani żadne słowo kluczowe
  else { Program odpowie „Nie zrozumiałem pytania. Zadaj pytanie z zakresu CI np. o nr zmiany jej status, błędy lub logi" Sugerując zawarcie w jednym pytaniu numeru oraz nazwę szukanych danych.

MySQLi + baza danych
Komunikacja z bazą danych w obie strony realizowana jest poprzez PHP + zapytania MySQLi.
Baza danych o nazwie **pomysl_cidb** dla uproszczenia projektu zawiera 2 tabele: tabelę **talk_history** na początku pustą ale w trakcie interakcji z aplikacją, zapełni się historią rozmów (pytaniami użytkownika i odpowiedziami ChatBota) Druga tabela **ci_table** przechowuje spreparowane dane na potrzeby zasymulowania interakcji z prawdziwą bazą danych CI.

- **talk_history -** klucz główny: id_talk z autouzupełnianiem numeru rekordu. Struktura:

#	Nazwa	Typ
1	id_talk	int(11)
2	talk	varchar(255)

Rys. Struktura tabeli **talk_history** widzianą z poziomu phpMyAdmin (dostępne pola i ich typy)

- **ci_table** - klucz główny: id_integration z autouzupełnianiem numeru rekordu. Struktura:

#	Nazwa	Typ
1	**id_integration**	int(11)
2	**status_integration**	varchar(255)
3	**error_integration**	varchar(255)
4	**log_integration**	varchar(255)
5	**user_integration**	varchar(255)

Rys. Struktura tabeli **ci_table** widzianą z poziomu phpMyAdmin (dostępne pola i ich typy)

id_integration	status_integration	error_integration	log_integration	user_integration
1	0 - waiting	0	integration:/index.html integration:/about.html	Marcin Krajewski
2	1 - in progress	0	integration:/css/style.css	Marek Semeniuk
3	3 - successful	0	integration:/index.php integration:/php/api.php	Mateusz Belau
4	4 - error	404 - the file could not be replaced	integration:/php/connect.php	Radosław Piotrowski
5	4 - error	505 - file is empty	integration:/js/var.js	Marcin Krajewski
6	4 - error	606 - db_user access denied	db_user:name_hacker	Marek Semeniuk
7	4 - error	707 - no url address	url:https://chatbot/db/	Mateusz Belau

Rys. przedstawiający rekordy spreparowanej bazy CI dla potrzeb realizacji warunków zadania

* Na potrzeby zadania, adresy widniejące w **log_integration** są fikcyjnymi adresami i te rekordy zawierają same w sobie całą informację o ”logach”. Aplikacja nie udostępnia wyświetlania zawartości urli.

Dane niezbędne do pracy z bazą:

Server:	localhost
Użytkownik / Admin:	pomysl_ciadm
Hasło:	321qaz
Nazwa bazy danych:	pomysl_cidb

Małe podsumowanie i wnioski

Analiza mechanizmu decyzyjnego przydzielania odpowiedzi ukazała nie spełnianie wszystkich zakładanych funkcjonalności.

Program świetnie rozpoznaje łączone pytania zawierające:

numer zmiany + dowolny rodzaj słowa kluczowego np.

JA: Chciałbym poznać błędy i logi z mojej ostatniej zmiany 6.
CI-BOT: Zmiana o numerze: 6, ma logi: db_user:name_hacker, ma błąd: 606 - db_user access denied
JA: Bardzo ale to bardzo martwię się o moją integrację 3, jaki ona ma status ?
CI-BOT: Zmiana o numerze: 3, ma status: 3 – successful

Ale

Analizując - scenariusz A:

Chatbot : Witaj, odpowiadam na pytania CI. Zadaj mi pytanie.
Użytkownik : Jaki jest status zmiany?
Chatbot : Podaj numer zmiany, która Cię interesuje:
Użytkownik: 1234
Chatbot : Zmiana 1234 jest w trakcie integracji

Program w przypadku podania poprawnego, istniejącego numeru zmiany w bazie CI ale bez żadnych słów kluczowych poda poprawnie status co jest logiczne i co spełnia warunki zadania na podstawie scenariusza A.
Ale już gdyby w scenariuszu A zamiast pytania o status było pytanie o logi lub błędy to podanie statusu zmiany było by nie poprawne.
(zaznaczono kod odpowiedzialny za tą decyzję na niebiesko w części opisującej mechanizm przydzielania odpowiedzi).

Analizując - scenariusz B:

Chatbot : Witaj, odpowiadam na pytania CI. Zadaj mi pytanie.
Użytkownik : Jaka jest przyczyna błędu integracji zmiany 5678?
Chatbot : Zmiana spowodowana przez błąd: brak komunikacji z serwerem. Czy chcesz zobaczyć logi?
Użytkownik: Tak
Chatbot : oto logi z testów buildu 5678

W założeniu odpowiedź „tak” powinna spowodować uzyskanie logów ale że odpowiedź „tak” nie spełnia warunków podstawowych stworzonego programu (nie zawiera informacji o numerze zmiany itd.) ChatBot zaproponuje: „Podaj numer zmiany, która Cię interesuje. Posiadam informacje o 7 zmianach. Najlepiej do numeru dopisz co chcesz wiedzieć o statusie, błędach lub logach. Przykład: Chciałbym poznać aktualny status oraz logi ze zmiany numer 3.”
Co powinno skłonić użytkownika do podania w jednym pytaniu numeru i słowa kluczowego logi, na co program ładnie mu odpowie.