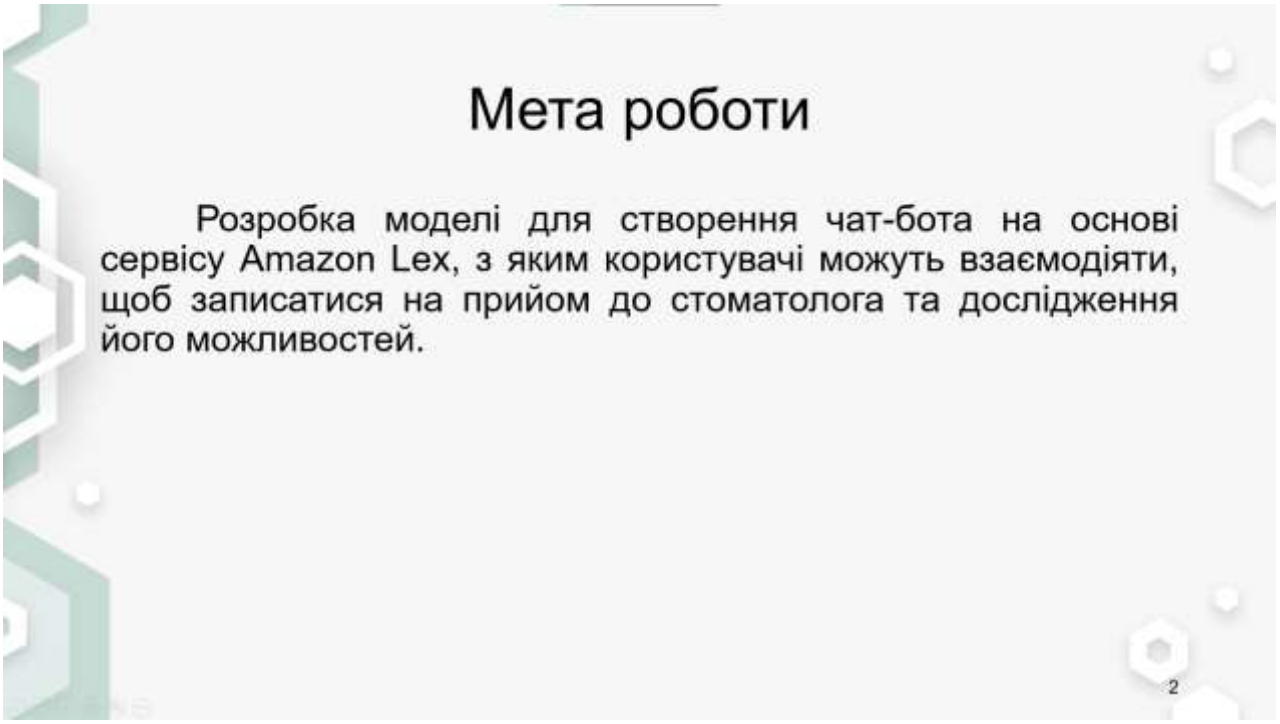
ДОДАТОК А
СЛАЙДИ ПРЕЗЕНТАЦІЇ



Зміст

Що таке Обробка природньої мови (Natural Language Processing)

Amazon Transcribe

Medical transcription

Subtitles

Streaming content labeling

Call center monitoring

5

Amazon Polly

News service production

Language training

Navigation systems

Animation production

6

Amazon Translate

International websites

Software localization

Multilingual chatbots

International media

7

Amazon Comprehend

Document analysis

Fraud detection

Mobile app analysis

Content management

8

Amazon Lex

Inventory and sales

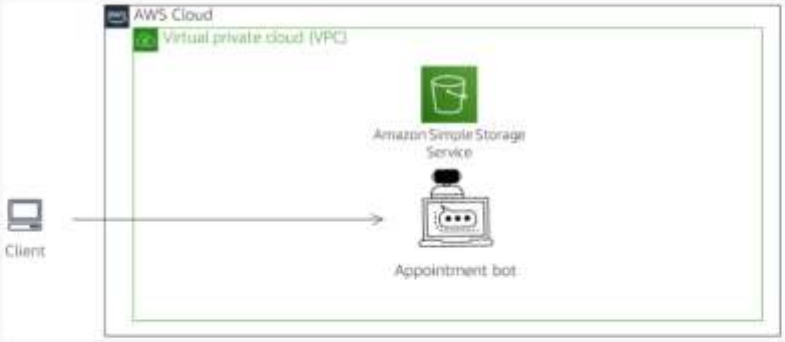
Interactive assistants

Customer service interfaces

Database queries

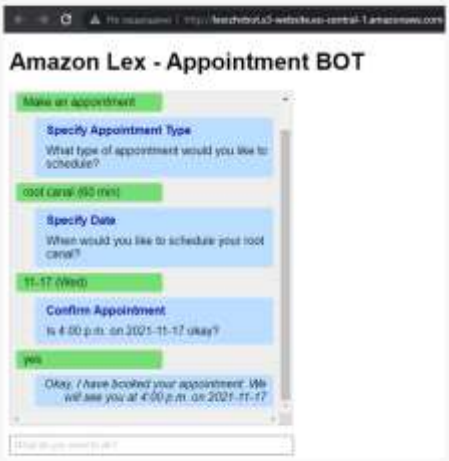
9

Модель середовища для чат-бота



10

Інтерфейс взаємодії з ботом



11

Висновки

- В даній дипломній роботі було розглянуто різноманітні сервіси для обробки природньої мови. Проаналізовано та наведено їх основні функції та випадки використання. Розглянуто проблеми та випадки використання обробки природньої мови.
- Було створено і протестовано бота за допомогою плану зустрічі Amazon Lex SheduleAppointment; функцію AWS Lambda для виконання завдань ініціалізації, перевірки та виконання. Налаштовано намір MakeAppointment для використання функції Lambda як гачка коду. Також було перевірено бота на консолі та через статичну веб-сторінку, розміщену на Amazon Simple Storage Service (Amazon S3).

12


Дякую за увагу!
13

ДОДАТОК Б

КОД

lambda_function.py

```
"""
 This code sample demonstrates an implementation of the Lex Code Hook Interface
 in order to serve a bot which manages dentist appointments.
 Bot, Intent, and Slot models which are compatible with this sample can be found in the Lex Console
 as part of the 'MakeAppointment' template.

 For instructions on how to set up and test this bot, as well as additional samples,
 visit the Lex Getting Started documentation http://docs.aws.amazon.com/lex/latest/dg/getting-started.html.
"""
import json
import dateutil.parser
import datetime
import time
import os
import math
import random
import logging
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
""" --- Helpers to build responses which match the structure of the necessary dialog actions --- """
def elicit_slot(session_attributes, intent_name, slots, slot_to_elicit, message, response_card):
    return {
        'sessionAttributes': session_attributes,
        'dialogAction': {
            'type': 'ElicitSlot',
            'intentName': intent_name,
            'slots': slots,
            'slotToElicit': slot_to_elicit,
            'message': message,
            'responseCard': response_card
        }
    }
def confirm_intent(session_attributes, intent_name, slots, message, response_card):
    return {
        'sessionAttributes': session_attributes,
        'dialogAction': {
            'type': 'ConfirmIntent',
            'intentName': intent_name,
            'slots': slots,
            'message': message,
            'responseCard': response_card
        }
    }
def close(session_attributes, fulfillment_state, message):
    response = {
        'sessionAttributes': session_attributes,
        'dialogAction': {
            'type': 'Close',
            'fulfillmentState': fulfillment_state,
            'message': message
        }
    }
    return response
def delegate(session_attributes, slots):
    return {
        'sessionAttributes': session_attributes,
        'dialogAction': {
            'type': 'Delegate',
            'slots': slots
        }
    }
def build_response_card(title, subtitle, options):
```

```
    """
    Build a responseCard with a title, subtitle, and an optional set of options which should be displayed as buttons.
    """
    buttons = None
    if options is not None:
        buttons = []
        for i in range(min(5, len(options))):
            buttons.append(options[i])
    return {
        'contentType': 'application/vnd.amazonaws.card.generic',
        'version': 1,
        'genericAttachments': [{
            'title': title,
            'subTitle': subtitle,
            'buttons': buttons
        }]
    }
""" --- Helper Functions --- """
def parse_int(n):
    try:
        return int(n)
    except ValueError:
        return float('nan')
def try_ex(func):
    """
    Call passed in function in try block. If KeyError is encountered return None.
    This function is intended to be used to safely access dictionary.
    Note that this function would have negative impact on performance.
    """
    try:
        return func()
    except KeyError:
        return None
def increment_time_by_thirty_mins(appointment_time):
    hour, minute = list(map(int, appointment_time.split(':')))
    return '{}:00'.format(hour + 1) if minute == 30 else '{}:30'.format(hour)
def get_random_int(minimum, maximum):
    """
    Returns a random integer between min (included) and max (excluded)
    """
    min_int = math.ceil(minimum)
    max_int = math.floor(maximum)
    return random.randint(min_int, max_int - 1)
def get_availabilities(date):
    """
    Helper function which in a full implementation would  feed into a backend API to provide query schedule availability.
    The output of this function is an array of 30 minute periods of availability, expressed in ISO-8601 time format.
    In order to enable quick demonstration of all possible conversation paths supported in this example, the function
    returns a mixture of fixed and randomized results.
    On Mondays, availability is randomized; otherwise there is no availability on Tuesday / Thursday and availability at
    10:00 - 10:30 and 4:00 - 5:00 on Wednesday / Friday.
    """
    day_of_week = dateutil.parser.parse(date).weekday()
    availabilities = []
    available_probability = 0.3
    if day_of_week == 0:
        start_hour = 10
        while start_hour <= 16:
            if random.random() < available_probability:
                # Add an availability window for the given hour, with duration determined by another random number.
                appointment_type = get_random_int(1, 4)
                if appointment_type == 1:
```

```
                availabilities.append('{}:00'.format(start_hour))
            elif appointment_type == 2:
                availabilities.append('{}:30'.format(start_hour))
            else:
                availabilities.append('{}:00'.format(start_hour))
                availabilities.append('{}:30'.format(start_hour))
        start_hour += 1
    if day_of_week == 2 or day_of_week == 4:
        availabilities.append('12:00')
        availabilities.append('16:00')
        availabilities.append('16:30')
    return availabilities
def isvalid_date(date):
    try:
        dateutil.parser.parse(date)
        return True
    except ValueError:
        return False
def is_available(appointment_time, duration, availabilities):
    """
    Helper function to check if the given time and duration fits within a known set of availability windows.
    Duration is assumed to be one of 30, 60 (meaning minutes).  Availabilities is expected to contain entries of the format HH:MM.
    """
    if duration == 30:
        return appointment_time in availabilities
    elif duration == 60:
        second_half_hour_time = increment_time_by_thirty_mins(appointment_time)
        return appointment_time in availabilities and second_half_hour_time in availabilities
    # Invalid duration ; throw error.  We should not have reached this branch due to earlier validation.
    raise Exception('Was not able to understand duration {}'.format(duration))
def get_duration(appointment_type):
    appointment_duration_map = {'cleaning': 30, 'root canal': 60, 'whitening': 30}
    return try_ex(lambda: appointment_duration_map[appointment_type.lower()])
def get_availabilities_for_duration(duration, availabilities):
    """
    Helper function to return the windows of availability of the given duration, when provided a set of 30 minute windows.
    """
    duration_availabilities = []
    start_time = '10:00'
    while start_time != '17:00':
        if start_time in availabilities:
            if duration == 30:
                duration_availabilities.append(start_time)
            elif increment_time_by_thirty_mins(start_time) in availabilities:
                duration_availabilities.append(start_time)
        start_time = increment_time_by_thirty_mins(start_time)
    return duration_availabilities
def build_validation_result(is_valid, violated_slot, message_content):
    return {
        'isValid': is_valid,
        'violatedSlot': violated_slot,
        'message': {'contentType': 'PlainText', 'content': message_content}
    }
def validate_book_appointment(appointment_type, date, appointment_time):
    if appointment_type and not get_duration(appointment_type):
        return build_validation_result(False, 'AppointmentType', 'I did not recognize that, can I book you a root canal, cleaning, or whitening?')
    if appointment_time:
        if len(appointment_time) != 5:
            return build_validation_result(False, 'Time', 'I did not recognize that, what time would you like to book your appointment?')
        hour, minute = appointment_time.split(':')
        hour = parse_int(hour)
        minute = parse_int(minute)
```

```
        if math.isnan(hour) or math.isnan(minute):
            return build_validation_result(False, 'Time', 'I did not recognize that, what time would you like to book your appointment?')
        if hour < 10 or hour > 16:
            # Outside of business hours
            return build_validation_result(False, 'Time', 'Our business hours are ten a.m. to five p.m.  What time works best for you?')
        if minute not in [30, 0]:
            # Must be booked on the hour or half hour
            return build_validation_result(False, 'Time', 'We schedule appointments every half hour, what time works best for you?')
    if date:
        if not isvalid_date(date):
            return build_validation_result(False, 'Date', 'I did not understand that, what date works best for you?')
        elif datetime.datetime.strptime(date, '%Y-%m-%d').date() <= datetime.date.today():
            return build_validation_result(False, 'Date', 'Appointments must be scheduled a day in advance.  Can you try a different date?')
        elif dateutil.parser.parse(date).weekday() == 5 or dateutil.parser.parse(date).weekday() == 6:
            return build_validation_result(False, 'Date', 'Our office is not open on the weekends, can you provide a work day?')
    return build_validation_result(True, None, None)
def build_time_output_string(appointment_time):
    hour, minute = appointment_time.split(':')  # no conversion to int in order to have original string form, for eg) 10:00 instead of 10:0
    if int(hour) > 12:
        return '{}:{} p.m.'.format((int(hour) - 12), minute)
    elif int(hour) == 12:
        return '12:{} p.m.'.format(minute)
    elif int(hour) == 0:
        return '12:{} a.m.'.format(minute)
    return '{}:{} a.m.'.format(hour, minute)
def build_available_time_string(availabilities):
    """
    Build a string eliciting for a possible time slot among at least two availabilities.
    """
    prefix = 'We have availabilities at '
    if len(availabilities) > 3:
        prefix = 'We have plenty of availability, including '
    prefix += build_time_output_string(availabilities[0])
    if len(availabilities) == 2:
        return '{} and {}'.format(prefix, build_time_output_string(availabilities[1]))
    return '{}, {} and {}'.format(prefix, build_time_output_string(availabilities[1]), build_time_output_string(availabilities[2]))
def build_options(slot, appointment_type, date, booking_map):
    """
    Build a list of potential options for a given slot, to be used in responseCard generation.
    """
    day_strings = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
    if slot == 'AppointmentType':
        return [
            {'text': 'cleaning (30 min)', 'value': 'cleaning'},
            {'text': 'root canal (60 min)', 'value': 'root canal'},
            {'text': 'whitening (30 min)', 'value': 'whitening'}
        ]
    elif slot == 'Date':
        # Return the next five weekdays.
        options = []
        potential_date = datetime.date.today()
        while len(options) < 5:
            potential_date = potential_date + datetime.timedelta(days=1)
            if potential_date.weekday() < 5:
                options.append({'text': '{}-{} ({})'.format((potential_date.month), potential_date.day, day_strings[potential_date.weekday()]),
                                'value': potential_date.strftime('%A, %B %d, %Y')})
        return options
    elif slot == 'Time':
        # Return the availabilities on the given date.
        if not appointment_type or not date:
            return None
        availabilities = try_ex(lambda: booking_map[date])
```

```
        if not availabilities:
            return None
        availabilities = get_availabilities_for_duration(get_duration(appointment_type), availabilities)
        if len(availabilities) == 0:
            return None
        options = []
        for i in range(min(len(availabilities), 5)):
            options.append({'text': build_time_output_string(availabilities[i]), 'value': build_time_output_string(availabilities[i])})
        return options
*** --- Functions that control the bot's behavior --- ***
def make_appointment(intent_request):
    """
    Performs dialog management and fulfillment for booking a dentists appointment.

    Beyond fulfillment, the implementation for this intent demonstrates the following:
    1) Use of elicitSlot in slot validation and re-prompting
    2) Use of confirmIntent to support the confirmation of inferred slot values, when confirmation is required
    on the bot model and the inferred slot values fully specify the intent.
    """
    appointment_type = intent_request['currentIntent']['slots']['AppointmentType']
    date = intent_request['currentIntent']['slots']['Date']
    appointment_time = intent_request['currentIntent']['slots']['Time']
    source = intent_request['invocationSource']
    output_session_attributes = intent_request['sessionAttributes'] if intent_request['sessionAttributes'] is not None else {}
    booking_map = json.loads(try_ex(lambda: output_session_attributes['bookingMap']) or '{}')
    if source == 'DialogCodeHook':
        # Perform basic validation on the supplied input slots.
        slots = intent_request['currentIntent']['slots']
        validation_result = validate_book_appointment(appointment_type, date, appointment_time)
        if not validation_result['isValid']:
            slots[validation_result['violatedSlot']] = None
            return elicit_slot(
                output_session_attributes,
                intent_request['currentIntent']['name'],
                slots,
                validation_result['violatedSlot'],
                validation_result['message'],
                build_response_card(
                    'Specify {}'.format(validation_result['violatedSlot']),
                    validation_result['message']['content'],
                    build_options(validation_result['violatedSlot'], appointment_type, date, booking_map)
                )
            )
        if not appointment_type:
            return elicit_slot(
                output_session_attributes,
                intent_request['currentIntent']['name'],
                intent_request['currentIntent']['slots'],
                'AppointmentType',
                {'contentType': 'PlainText', 'content': 'What type of appointment would you like to schedule?'},
                build_response_card(
                    'Specify Appointment Type', 'What type of appointment would you like to schedule?',
                    build_options('AppointmentType', appointment_type, date, None)
                )
            )
        if appointment_type and not date:
            return elicit_slot(
                output_session_attributes,
                intent_request['currentIntent']['name'],
                intent_request['currentIntent']['slots'],
                'Date',
                {'contentType': 'PlainText', 'content': 'When would you like to schedule your {}?'.format(appointment_type)},
```

```
                build_response_card(
                    'Specify Date',
                    'When would you like to schedule your {}?'.format(appointment_type),
                    build_options('Date', appointment_type, date, None)
                )
            )
        if appointment_type and date:
            # Fetch or generate the availabilities for the given date.
            booking_availabilities = try_ex(lambda: booking_map[date])
            if booking_availabilities is None:
                booking_availabilities = get_availabilities(date)
                booking_map[date] = booking_availabilities
                output_session_attributes['bookingMap'] = json.dumps(booking_map)
            appointment_type_availabilities = get_availabilities_for_duration(get_duration(appointment_type), booking_availabilities)
            if len(appointment_type_availabilities) == 0:
                # No availability on this day at all; ask for a new date and time.
                slots['Date'] = None
                slots['Time'] = None
                return elicit_slot(
                    output_session_attributes,
                    intent_request['currentIntent']['name'],
                    slots,
                    'Date',
                    {'contentType': 'PlainText', 'content': 'We do not have any availability on that date, is there another day which works for you?'},
                    build_response_card(
                        'Specify Date',
                        'What day works best for you?',
                        build_options('Date', appointment_type, date, booking_map)
                    )
                )
            message_content = 'What time on {} works for you? '.format(date)
            if appointment_time:
                output_session_attributes['formattedTime'] = build_time_output_string(appointment_time)
                # Validate that proposed time for the appointment can be booked by first fetching the availabilities for the given day.  To
                # give consistent behavior in the sample, this is stored in sessionAttributes after the first lookup.
                if is_available(appointment_time, get_duration(appointment_type), booking_availabilities):
                    return delegate(output_session_attributes, slots)
                message_content = 'The time you requested is not available. '
            if len(appointment_type_availabilities) == 1:
                # If there is only one availability on the given date, try to confirm it.
                slots['Time'] = appointment_type_availabilities[0]
                return confirm_intent(
                    output_session_attributes,
                    intent_request['currentIntent']['name'],
                    slots,
                    {
                        'contentType': 'PlainText',
                        'content': '{}{} is our only availability, does that work for you?'.format
                                   (message_content, build_time_output_string(appointment_type_availabilities[0]))
                    },
                    build_response_card(
                        'Confirm Appointment',
                        'Is {} on {} okay?'.format(build_time_output_string(appointment_type_availabilities[0]), date),
                        [{'text': 'yes', 'value': 'yes'}, {'text': 'no', 'value': 'no'}]
                    )
                )
            available_time_string = build_available_time_string(appointment_type_availabilities)
            return elicit_slot(
                output_session_attributes,
                intent_request['currentIntent']['name'],
                slots,
                'Time',
                {'contentType': 'PlainText', 'content': '{}{}'.format(message_content, available_time_string)},

                build_response_card(
                    'Specify Time',
                    'What time works best for you?',
                    build_options('Time', appointment_type, date, booking_map)
                )
            )
        return delegate(output_session_attributes, slots)
    # Book the appointment.  In a real bot, this would likely involve a call to a backend service.
    duration = get_duration(appointment_type)
    booking_availabilities = booking_map[date]
    if booking_availabilities:
        # Remove the availability slot for the given date as it has now been booked.
        booking_availabilities.remove(appointment_time)
        if duration == 60:
            second_half_hour_time = increment_time_by_thirty_mins(appointment_time)
            booking_availabilities.remove(second_half_hour_time)
        booking_map[date] = booking_availabilities
        output_session_attributes['bookingMap'] = json.dumps(booking_map)
    else:
        # This is not treated as an error as this code sample supports functionality either as fulfillment or dialog code hook.
        logger.debug('Availabilities for {} were null at fulfillment time.  '
                     'This should have been initialized if this function was configured as the dialog code hook'.format(date))
    return close(
        output_session_attributes,
        'Fulfilled',
        {
            'contentType': 'PlainText',
            'content': 'Okay, I have booked your appointment.  We will see you at {} on {}'.format(build_time_output_string(appointment_time), date)
        }
    )


""" --- Intents --- """
def dispatch(intent_request):
    """
    Called when the user specifies an intent for this bot.
    """
    logger.debug('dispatch userId={}, intentName={}'.format(intent_request['userId'], intent_request['currentIntent']['name']))
    intent_name = intent_request['currentIntent']['name']
    # Dispatch to your bot's intent handlers
    if intent_name == 'MakeAppointment':
        return make_appointment(intent_request)
    raise Exception('Intent with name ' + intent_name + ' not supported')


""" --- Main handler --- """
def lambda_handler(event, context):
    """
    Route the incoming request based on intent.
    The JSON body of the request is provided in the event slot.
    """
    # By default, treat the user request as coming from the America/New_York time zone.
    os.environ['TZ'] = 'America/New_York'
    time.tzset()
    logger.debug('event.bot.name={}'.format(event['bot']['name']))
    return dispatch(event)
```

index.html

index.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Amazon Lex for JavaScript - Sample Application (NTBBot)</title>
    <script src="https://sdk.amazonaws.com/js/aws-sdk-2.633.0.min.js"></script>
    <style language="text/css">
        body {
            font-family:  "Helvetica Neue", "Arial";
        }

        input#wisdom {
            padding: 4px;
            font-size: 1em;
            width: 380px
        }

        input::placeholder {
            color: #ccc;
            font-style: italic;
        }

        p.userRequest {
            margin: 4px;
            padding: 4px 10px 4px 10px;
            border-radius: 4px;
            min-width: 50%;
            max-width: 85%;
            float: left;
            background-color: #7d7;
        }

        p.lexResponse {
            margin: 4px;
            padding: 4px 10px 4px 10px;
            border-radius: 4px;
            text-align: right;
            min-width: 50%;
            max-width: 85%;
            float: right;
            background-color: #bdf;
            font-style: italic;
        }

        span.cardTitle {
            display: block;
            float: left;
            width:  100%;
            margin: 0px;
            padding: 5px;
            text-align: left;
            font-style: normal;
            font-weight: bolder;
            color: #009
        }

```

```
57          span.cardSubtitle {
58              display: block;
59              float: left;
60              width:  100%;
61              margin: 0px;
62              padding: 5px;
63              text-align: left;
64              font-style: normal;
65              font-weight: normal
66          }
67
68          span.cardOptions {
69              display: block;
70              float: left;
71              width:  90%;
72              margin: 0px;
73              padding: 12px;
74              text-align: center
75          }
76
77          a.cardOption {
78              display:  inline-block;
79              margin: 4px 8px 4px 8px;
80              padding: 6px;
81              border-radius: 6px;
82              background-color: #eef;
83              color: #009;
84              text-align: center;
85              font-style: normal;
86              font-weight: normal;
87              text-decoration: none
88          }
89
90          a.cardOption:hover {
91              background-color:  #ccf;
92          }
93
94          p.lexError {
95              margin: 4px;
96              padding: 4px 10px 4px 10px;
97              border-radius: 4px;
98              text-align: right;
99              min-width: 50%;
100             max-width: 85%;
101             float: right;
102             background-color: #f77;
103         }
104
105         a.appAction {
106             width:  80px;
107             display:  inline-block;
108             margin: 0px 8px 4px 8px;
109             padding: 12px;
110             border-radius: 6px;
111             background-color: #f67e00;
112             color: #fff;
113             text-align: center;
```

```
            font-style: normal;
            font-weight: normal;
            text-decoration: none
        }

        a.appAction:hover {
            background-color:  #e66e00;;
        }
    </style>
    <meta name="viewport" content="width=420">
</head>

<body>
    <h1 style="text-align:  left">Amazon Lex - Appointment BOT</h1>
    <div id="searchResults" style="width: 380px; height: 450px; border: 1px solid #ccc; background-color: #eee;
    padding: 4px; overflow: scroll; margin-right: 10px; float: left; display: none"></div>
    <div id="conversation" style="width: 380px; height: 450px; border: 1px solid #ccc; background-color: #eee;
    padding: 4px; overflow: scroll"></div>
    <form id="chatform" style="margin-bottom: 10px" onsubmit="return pushChat();">
        <p style="width: 380px; font-size: 0.8em; line-height: 1.2em">
            <input type="text" id="wisdom" size="80" value="" placeholder="What do you want to do?">
        </p>
    </form>
    <script type="text/javascript">
        // set the focus to the input box
        document.getElementById("wisdom").focus();

        // Initialize the Amazon Cognito credentials provider
        AWS.config.region = 'us-east-1'; // Region
        AWS.config.credentials = new AWS.CognitoIdentityCredentials({
            IdentityPoolId: 'us-east-1:a198ff70-35e5-4b99-b9c5-49ed41197a47',
        });

        var lexruntime = new AWS.LexRuntime();
        var lexUserId = 'chatbot-demo' + Date.now();
        var sessionAttributes = {};
        var responseCardOptions = null;

        function pushChat(textToPush, labelToShow) {

            // if there is text to be sent...
            var wisdomText = document.getElementById('wisdom');
            if (wisdomText && wisdomText.value && wisdomText.value.trim().length > 0) {

                // disable input to show we're sending it
                var wisdom = wisdomText.value.trim();
                wisdomText.value = '...';
                wisdomText.locked = true;

                // if a response card was being used and the user typed in an option that is one
                // of the responses, then send the value of the response, not the text that was typed in
                if (responseCardOptions && (! labelToShow)) {

                    // this means a response card was shown to the user, but she chose to type in a response instead
                    // of clicking
                    // TODO:  hide the responseCardOptions
```

```
                for (var i=0; (i < responseCardOptions.length); (i++)) {

                    var item = responseCardOptions[i];
                    if (wisdom.toUpperCase() === item.text.toUpperCase()) {

                        textToPush = item.value;
                        labelToShow = item.text;
                        // break out of the loop
                        i = responseCardOptions.length + 1;
                    }
                }
            }

            // send it to the Lex runtime
            var params = {
                botAlias: 'rzhvbot',
                botName: 'ScheduleAppointment',
                // if the caller has specifically provided text to be pushed, use it
                inputText: (textToPush ? textToPush : wisdom),
                userId: lexUserId,
                sessionAttributes: sessionAttributes
            };

            // if the caller has specifically provided a label to be shown, show it
            showRequest((labelToShow ? labelToShow : wisdom));

            // do a KB search if requested (not used when the input is from a response card)
            /*
            if (params.inputText != textToPush) {
                showKBResponse(params.inputText);
            }
            */

            lexruntime.postText(params, function(err, data) {

                if (err) {
                    console.log(err, err.stack);
                    showError('Error:  ' + err.message + ' (see console for details)')
                }
                if (data) {

                    // check for missed utterances
                    checkForMissedUtterances(data);

                    // capture the sessionAttributes for the next cycle
                    sessionAttributes = data.sessionAttributes;
                    // reset the responseCardOptions
                    responseCardOptions = null;
                    // show response and/or error/dialog status
                    showResponse(data);
                }
                // re-enable input
                wisdomText.value = '';
                wisdomText.locked = false;
            });
        }
        // we always cancel form submission
```

```
        return false;
    }

    function checkForMissedUtterances(lexResponse) {

        // clarificationPrompt entries for the bot so we know when we get these responses
        var clarificationPrompts = [
            'I hate to admit it, but I only know about Amazon Lex FAQs at this time. Sorry.',
            'Sorry; I don\'t understand that. Can we try something easier like, "Tell me about Amazon Lex."'
        ];

        if (! lexResponse.sessionAttributes) {

            lexResponse.sessionAttributes = {};
            lexResponse.sessionAttributes.missedUtterances = '0';
        }

        // missed utterances result in an ElicitIntent
        if (lexResponse.dialogState && (lexResponse.dialogState === 'ElicitIntent')) {

            // missed utterances result in clarificationPrompts
            if (lexResponse.message && (clarificationPrompts.indexOf(lexResponse.message) != -1)) {

                lexResponse.sessionAttributes.missedUtterances =
                    ((parseInt(lexResponse.sessionAttributes.missedUtterances) || 0) + 1).toString();
                return;
            }
        }

        // if we got to this point, then we didn't have a missed utterance
        lexResponse.sessionAttributes.missedUtterances = '0';
    }

    function showRequest(daText) {

        var conversationDiv = document.getElementById('conversation');
        var requestPara = document.createElement("P");
        requestPara.className = 'userRequest';
        requestPara.appendChild(document.createTextNode(daText));
        conversationDiv.appendChild(requestPara);
        conversationDiv.scrollTop = conversationDiv.scrollHeight;

        // also add it to the kbDiv
        var kbDiv = document.getElementById('searchResults');
        requestPara = document.createElement("P");
        requestPara.className = 'userRequest';
        requestPara.appendChild(document.createTextNode(daText));
        kbDiv.appendChild(requestPara);
        kbDiv.scrollTop = kbDiv.scrollHeight;
    }

    function showError(daText) {

        var conversationDiv = document.getElementById('conversation');
        var errorPara = document.createElement("P");
        errorPara.className = 'lexError';
        errorPara.appendChild(document.createTextNode(daText));
```

```
284             conversationDiv.appendChild(errorPara);
285             conversationDiv.scrollTop = conversationDiv.scrollHeight;
286         }
287
288         function postCardOption(optionLabel, optionValue, optionsSpan) {
289
290             // hide the options, now that one has been chosen
291             // TODO:  also do this if the user has typed an answer
292             optionsSpan.style.display = 'none';
293
294             // post the optionLabel to the bot, but show the optonValue
295             var wisdomText = document.getElementById('wisdom');
296             if (wisdomText) {
297
298                 wisdomText.value = optionLabel;
299                 pushChat(optionValue, optionLabel);
300             }
301         }
302
303         function renderResponseCard(responseCard, responsePara) {
304
305             if (responseCard.genericAttachments && responseCard.genericAttachments.length > 0) {
306
307                 // we will render at most 1 card
308                 var card = responseCard.genericAttachments[0];
309
310                 // title
311                 var elem = document.createElement('span');
312                 elem.className = 'cardTitle';
313                 elem.appendChild(document.createTextNode(card.title));
314                 responsePara.appendChild(elem);
315
316                 // subtitle
317                 elem = document.createElement('span');
318                 elem.className = 'cardSubtitle';
319                 // should we treat the subtitle as text or as HTML?
320                 // elem.appendChild(document.createTextNode(card.subTitle));
321                 elem.innerHTML = card.subTitle;
322                 responsePara.appendChild(elem);
323
324                 // options
325                 if (card.buttons && (card.buttons.length > 0)) {
326
327                     var optionsSpan = document.createElement('span');
328                     optionsSpan.className = 'cardOptions';
329                     responsePara.appendChild(optionsSpan);
330
331                     // remember the responseCardOptions so we know if the user types one in
332                     responseCardOptions = card.buttons;
333
334                     for (var i=0; (i < card.buttons.length); i++) {
335
336                         var item = card.buttons[i];
337
338                         elem = document.createElement('a');
339                         elem.className = 'cardOption';
340                         elem.href = '#';
```

```
                    elem.setAttribute('onclick', 'postCardOption("${item.text}", "${item.value}", this.parentNode); return false;');
                    elem.title = item.text;
                    elem.appendChild(document.createTextNode(item.text));

                    optionsSpan.appendChild(elem);
                }
            }
        } else
            throw new Error('I have no response card content to work with');
    }

    function showResponse(lexResponse) {

        var conversationDiv = document.getElementById('conversation');
        var responsePara = document.createElement("P");
        responsePara.className = 'lexResponse';

        // look for a response card first
        if (lexResponse.responseCard) {

            // if a larger title is available, use it
            if (lexResponse.sessionAttributes && lexResponse.sessionAttributes.fullTitle &&
                (lexResponse.sessionAttributes.fullTitle != lexResponse.responseCard.genericAttachments[0].title)) {

                lexResponse.responseCard.genericAttachments[0].title = lexResponse.sessionAttributes.fullTitle;
            }

            // if a larger subtitle is available, use it
            if (lexResponse.sessionAttributes && lexResponse.sessionAttributes.fullSubtitle &&
                (lexResponse.sessionAttributes.fullSubtitle != lexResponse.responseCard.genericAttachments[0].subTitle)) {

                lexResponse.responseCard.genericAttachments[0].subTitle = lexResponse.sessionAttributes.fullSubtitle;
            }
            renderResponseCard(lexResponse.responseCard, responsePara);

        } else if (lexResponse.message) {
            responsePara.appendChild(document.createTextNode(lexResponse.message));
            responsePara.appendChild(document.createElement('br'));
        }
        /* don't show dialog state for now
        if (lexResponse.dialogState === 'ReadyForFulfillment') {
            responsePara.appendChild(document.createTextNode('Ready for fulfillment'));
            // TODO:  show slot values
        } else {

            missedUtterances = lexResponse.sessionAttributes.missedUtterances;
            responsePara.appendChild(document.createTextNode(
                '(' + lexResponse.dialogState +
                    ((missedUtterances !== '0') ? '; ' + missedUtterances + ' missed' : '') + ')'));
        }
        */
        conversationDiv.appendChild(responsePara);
        conversationDiv.scrollTop = conversationDiv.scrollHeight;
    }

    function showKBResponse(queryPhrase) {
```

```
            if (document.getElementById('showSearchResults').checked) {

                // add to the other div
                var kbDiv = document.getElementById('searchResults');
                if (kbDiv.style.display !== 'block') {
                    kbDiv.style.display = 'block';
                }
                var responsePara = document.createElement("P");
                responsePara.className = 'lexResponse';

                // run the query and get the response (asynchronously)
                loadResults(queryPhrase, responsePara);

                kbDiv.appendChild(responsePara);
                kbDiv.scrollTop = kbDiv.scrollHeight;
            }
        }

        function loadResults(queryPhrase, responsePara) {

            var xhttp = new XMLHttpRequest();
            xhttp.onreadystatechange = function() {

                if (this.readyState == 4 && this.status == 200) {

                    var searchResponse = JSON.parse(this.responseText);
                    if (searchResponse.hits.total && (searchResponse.hits.total > 0)) {

                        // create an element so we can use HTML here...
                        // title
                        var elem = document.createElement('span');
                        elem.className = 'cardTitle';
                        elem.innerHTML = searchResponse.hits.hits[0]._source.question;
                        responsePara.appendChild(elem);
                        responsePara.appendChild(document.createElement('br'));

                        // subtitle
                        elem = document.createElement('span');
                        elem.className = 'cardSubtitle';
                        elem.innerHTML = searchResponse.hits.hits[0]._source.answer;
                        responsePara.appendChild(elem);
                        responsePara.appendChild(document.createElement('br'));
                    }
                    responsePara.appendChild(document.createTextNode(`${searchResponse.hits.total} hit(s)`));

                    // scroll to the bottom
                    var kbDiv = document.getElementById('searchResults');
                    kbDiv.scrollTop = kbDiv.scrollHeight;
                }
            };

            xhttp.open('GET',
                'https://s7qryxr105.execute-api.us-east-2.amazonaws.com/QA/?queryPhrase=' +
                    encodeURIComponent(queryPhrase),
                true);
            xhttp.send();
        }
```

```
        showResponse({
            message:  'Hi there!  What do you want to do?'
        });


    </script>
</body>

</html>
```

error.html

error.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Amazon Lex for JavaScript - Sample Application (NTBBot)</title>
    <script src="https://sdk.amazonaws.com/js/aws-sdk-2.633.0.min.js"></script>
    <style language="text/css">
        body {
            font-family:  "Helvetica Neue", "Arial";
        }

        input#wisdom {
            padding: 4px;
            font-size: 1em;
            width: 380px
        }

        input::placeholder {
            color: #ccc;
            font-style: italic;
        }

        p.userRequest {
            margin: 4px;
            padding: 4px 10px 4px 10px;
            border-radius: 4px;
            min-width: 50%;
            max-width: 85%;
            float: left;
            background-color: #7d7;
        }

        p.lexResponse {
            margin: 4px;
            padding: 4px 10px 4px 10px;
            border-radius: 4px;
            text-align: right;
            min-width: 50%;
            max-width: 85%;
            float: right;
            background-color: #bdf;
            font-style: italic;
        }

        span.cardTitle {
            display: block;
            float: left;
            width:  100%;
            margin: 0px;
            padding: 5px;
            text-align: left;
            font-style: normal;
            font-weight: bolder;
            color: #009
        }

```

```
span.cardSubtitle {
    display: block;
    float: left;
    width:  100%;
    margin: 0px;
    padding: 5px;
    text-align: left;
    font-style: normal;
    font-weight: normal
}

span.cardOptions {
    display: block;
    float: left;
    width:  90%;
    margin: 0px;
    padding: 12px;
    text-align: center
}

a.cardOption {
    display:  inline-block;
    margin: 4px 8px 4px 8px;
    padding: 6px;
    border-radius: 6px;
    background-color: #eef;
    color: #009;
    text-align: center;
    font-style: normal;
    font-weight: normal;
    text-decoration: none
}

a.cardOption:hover {
    background-color:  #ccf;
}

p.lexError {
    margin: 4px;
    padding: 4px 10px 4px 10px;
    border-radius: 4px;
    text-align: right;
    min-width: 50%;
    max-width: 85%;
    float: right;
    background-color: #f77;
}

a.appAction {
    width:  80px;
    display:  inline-block;
    margin: 0px 8px 4px 8px;
    padding: 12px;
    border-radius: 6px;
    background-color: #f67e00;
    color: #fff;
```

```
            text-align: center;
            font-style: normal;
            font-weight: normal;
            text-decoration: none
        }

        a.appAction:hover {
            background-color:  #e66e00;;
        }
    </style>
    <meta name="viewport" content="width=420">
</head>

<body>
    <h1 style="text-align:  left">Amazon Lex - Appointment BOT error</h1>

    <div id="searchResults" style="width: 380px; height: 450px; border: 1px solid #ccc; background-color: #eee;
    padding: 4px; overflow: scroll; margin-right: 10px; float: left; display: none"></div>
    <div id="conversation" style="width: 380px; height: 450px; border: 1px solid #ccc; background-color: #eee;
    padding: 4px; overflow: scroll"></div>
    <form id="chatform" style="margin-bottom: 10px" onsubmit="return pushChat();">
        <p style="width: 380px; font-size: 0.8em; line-height: 1.2em">
            <input type="text" id="wisdom" size="80" value="" placeholder="What do you want to do?">
        </p>
    </form>

     <h2 style="text-align: left">There is an error</h2>
</body>

</html>
```

ДОДАТОК В
ПУБЛІКАЦІЇ ЗА ТЕМОЮ РОБОТИ

Editorial board of International Electronic Scientific and Practical Journal «WayScience» (ISSN 2664-4819 (Online)

Розвиток освіти, науки та бізнесу: результати 2021: тези доп. міжнародної науково-практичної інтернет-конференції, 6-7 грудня 2021 р. – Дніпро, Україна, 2021. – 168 с.

(Development of Education, Science and Business: Results 2021: abstracts of the International Scientific and Practical Internet Conference, December 6-7, 2021. – Dnipro, Ukraine, 2021. – 168 p.)

International Scientific and Practical Internet Conference "Development of Education, Science and Business: Results 2021" devoted to the main research of this year.

Topics cover all sections of the International Electronic Scientific and Practical Journal "WayScience", namely:

- public administration;
- philosophical sciences;
- economic sciences;
- historical sciences;
- legal sciences;
- agricultural sciences;
- geographic sciences;
- pedagogical sciences;
- psychological sciences;
- sociological sciences;
- political sciences;
- philological sciences;
- technical sciences;
- medical sciences;
- chemical sciences;
- biological sciences;
- physical and mathematical sciences;
- other professional sciences.

Dnipro, Ukraine – 2021

AMAZON LEX – СТВОРЕННЯ ЧАТ-БОТА, ОБРОБКА ПРИРОДНОЇ МОВИ

Рижов О.О.
магістр, E-mail: oleksandr.ryzhov@nure.ua
Кривенко С.А.
кандидат технічних наук, доцент
Кафедра Інформаційно-мережної інженерії
Харківський національний університет радіоелектроніки
(61166, Харків, просп. Науки, 14, тел. (057) 702-14-29)

На сьогоднішній день обробка природної мови (NLP) є основною темою, яка досліджується та розробляється останнє десятиліття. Здатність аналізувати, обчислювати та розуміти просту людську мову за допомогою комп'ютерів і машинного навчання була впроваджена в різні програми та платформи. У цій роботі розглядається, як чат-боти в основному використовують такі параметри, як сутності та наміри, щоб узгоджувати синтаксис і виконувати обробку природною мовою. Поява віртуальних помічників у споживчих технологіях змусила великі компанії працювати на власних хмарних платформах.

Завдяки технології штучного інтелекту, що розвивається, останнім часом чат-боти стають розумнішими та швидшими. Чат-боти, як правило, доступні цілодобово, забезпечуючи постійну підтримку та послуги. Проблема розробки інтелектуального чат-бота все ще існує з моменту появи штучного інтелекту. Функціональність чат-ботів може варіюватися від коротких розмов, орієнтованих на бізнес, до триваліших розмов на основі медичних заходів. Однак головна роль, яку повинні відігравати чат-боти, полягає в розумінні людських висловлювань, щоб правильно реагувати.

У першому розділі розглядається обробка природньої мови, її проблеми та випадки використання. Обробка природної мови також відома як NLP (Natural Language Processing). NLP розробляє обчислювальні алгоритми для автоматичного аналізу та представлення людської мови. Оцінюючи структуру мови, системи машинного навчання ML можуть обробляти великі набори слів, фраз і речень. Машинне навчання ML – це підмножина більш широкої галузі інформатики, яка відома як штучний інтелект AI. AI – це створення будівельних машин, які можуть виконувати завдання, які зазвичай виконує людина. У сучасній культурі штучний інтелект з'являється у фільмах або художніх творах. Ви можете згадати деякі приклади штучного інтелекту у науково-фантастичних фільмах або телешоу, які контролюють майбутній світ або діють розумно самостійно - іноді з негативними наслідками для суспільства або оточуючих людей. Ці застосування AI починалися як комп'ютерні агенти, які сприймали їхнє середовище та вживали заходів для досягнення певної мети. Однак, для деяких із цих вигаданих прикладів штучного інтелекту їх дії не були результатом, якого спочатку передбачали їх творці. Інші вигадані приклади штучного інтелекту більш доброякісні або позитивні: вони краще працюють з людством, але вони також мають більш загальний характер. Такі види загального штучного інтелекту є прикладами сильного штучного загального інтелекту AGI. Вони мають здатність вивчати або розуміти будь-яке завдання, яке може зрозуміти людина.

Проблеми штучного інтелекту зазвичай охоплюють багато сфер дослідження: обробка природної мови, міркування, представлення знань, навчання, сприйняття та взаємодія з фізичним середовищем. AI ще не є реальністю, якщо ви не живете в симуляції. Однак з кожним роком це стає ближчим у кожній із цих областей.

Можливо, ви також читали чи бачили коментарі до етики створення AI. Не всі погляди позитивні – можливо частково через страх перед вигаданими штучними інтелектами, які знищили людей або використовували їх як джерела енергії. Деякі також

можуть розглянути ризик масового безробіття, оскільки розумна машина може працювати безперервно.

Обробка природної мови NLP – це ще одна сфера машинного навчання із збільшенням її використання. Наприклад, Amazon Alexa (або будь -який інший голосовий помічник) використовує NLP, щоб спробувати відповісти на ваші запитання. NLP – це мова, а також письмовий текст.

NLP використовується в багатьох додатках, таких як:

Боти чату або колл-центру – автоматизовані системи для отримання балансу в банку або замовлення їжі в ресторані.

Інструменти перекладу – перетворення тексту між мовами або програми, які можуть перекладати меню в режимі реального часу.

Переклад голосу в текст – перетворення вимовлених слів у текст. Може використовуватися для включення автоматичних субтитрів.

Аналіз настроїв – дозволяє аналізувати почуття коментарів у оглядах продуктів, музики та фільмів. Ці почуття можна використати, щоб оцінити аудиторію фільму.

NLP – це широкий термін для загального набору ділових або обчислювальних проблем, які можна вирішити за допомогою машинного навчання.

Після аналізу NLP в розділі розглядається попередня обробка тексту.

У другому розділі аналізуються досягнення компанії Amazon, розглядаються основні сервіси, які використовуються для обробки природньої мови. Amazon Transcribe може автоматично перетворювати розмовну мову на текст, його використовують для медичної транскрипції, субтитрів для відео, маркування потокового вмісту та для моніторингу колл-центрів. Amazon Polly може перетворювати письмовий текст у розмовну мову, наприклад, для виробництва служб новин, систем навчання мові, систем навігації та виробництва анімацій. Amazon Translate може створювати переклади між мовами в режимі реального часу, який можна використовувати для створення міжнародних веб-сайтів, локалізації програмного забезпечення, розробки багатомовних чат-ботів та міжнародного управління медіа. Amazon Comprehend автоматизує багато випадків використання NLP, які розглядаються в цій роботі, наприклад, для аналізу юридичних та медичних документів, виявлення фінансового шахрайства, масштабного аналізу мобільних додатків або позначення вмісту для аналізу та керування. І, нарешті, Amazon Lex може створити інтерфейс, схожий на людину, який може взаємодіяти за допомогою голосу та тексту, щоб задавати запитання, отримувати відповіді чи виконувати завдання, наприклад, створення інтерфейсів для управління запасами та програм продажів, інтерактивної допомоги, обслуговування клієнтів або запитів до баз даних людською мовою.

У третьому розділі розглядається синтез моделі середовища для роботи чат-бота з обробкою природньої мови на основі Amazon Lex. Описано виконання наступних кроків високого рівня: створення і тестування бота за допомогою плану зустрічі Amazon Lex ScheduleAppointment; створення і тестування функцію AWS Lambda для виконання завдань ініціації, перевірки та виконання; налаштування наміру MakeAppointment для використання функції Lambda як гачка коду. Кожен бот, який створюється в Amazon Lex, має намір. Намір – це дія, яку виконує бот.

Четвертий розділ присвячений тестуванню моделі для чат-бота, в тому числі, за допомогою веб-сторінки розміщеної на Amazon S3. Розглядається зібрання Amazon Lex та Lambda в одне середовище, перевірення бота, для доведення, що він виконує лямбда-функцію. Далі, тестується публікація бота, щоб можна було перевірити його виклик з веб-сторінки. Найпростіший спосіб провести тест – створити статичну веб-сторінку та розмістити її на Amazon S3. Перевіряється чи викликається API Amazon Lex з веб-сторінки для завантаження нашого бота, завдяки чому відвідувачі веб-сторінки можуть потім взаємодіяти з нею. Тестується налаштування пулу ідентифікаційних даних Amazon Cognito та створення відповідних ролей (одна роль закінчується на Auth_Role, а інша на Unauth_Role), для забезпечення безпеки на цій сторінці,

Веб-сторінці, на якій розміщено нашого бота, потрібен дозвіл на доступ до Amazon Lex. Для цього двом ролям, створеним у пулі ідентифікаційних даних Amazon Cognito, потрібні дозволи на доступ до Amazon Lex. Перевіряється налаштування AWS Identity and Access Management (IAM), щоб надати ці дозволи ролям пулу ідентифікаційних даних. Після налаштування дозволів безпеки і створення сегмента S3 для розміщення нашої веб-сторінки, тестується оновлений демонстраційний файл HTML, і перевіряється щоб він використовував пул ідентифікаційних даних Amazon Cognito. Фінальне тестування перевіряє взаємодію зі створеним в магістерській роботі ботом. На сторінці з розділом Статичний хостинг веб-сайтів за вибраним URL-адресом відкривається створена веб-сторінка.

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	*96 с.*
	Графічні документи	
2.	Слайди презентації	*13*

Змін.	Арк.	Номер докум.	Підпис	Дата	Amazon Lex – створення чат-бота, обробка природної мови	Літ.	Аркуш	Аркушів
Розроб.		Рижов						1
Перевір.		Кривенко						
						ХНУРЕ кафедра ІМІ		
Н. контр.		Кривенко						
Затв.		Безрук			Відомість кваліфікаційної роботи			