



SERVERLESS MESSENGER BOT WORKSHOP

Mikael Puittinen, Chief Technology Officer
Eetu Tuomala, Cloud Application Architect

16.11.2016

SC5 BRIEFLY



CLOUD
SOLUTIONS



BUSINESS
APPLICATIONS



INTELLIGENT
APPLICATIONS



DIGITAL
DESIGN

10
YEARS

60+
CUSTOMERS

200+
PROJECTS

85
HACKERS
DESIGNERS

**HEL
JKL**

~7
MEUR
2016 (FC)



A-lehdet



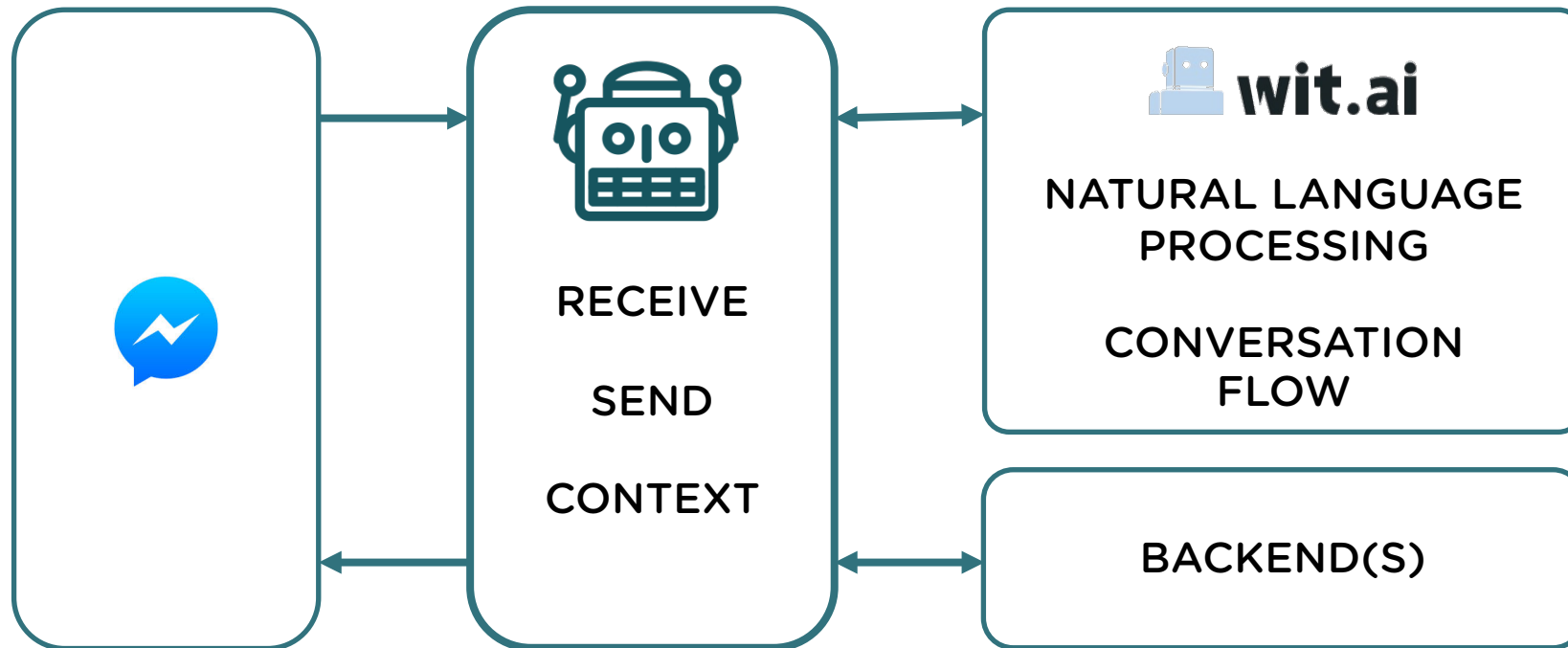
VISIT OUR WEB SITE FOR MORE INFO: [HTTPS://SC5.IO](https://sc5.io)

INTRODUCTION / PRE-REQUISITES

- Introduction to AWS & Serverless Framework :
<http://serverless.fi/docs/aws-serverless-intro.pdf>
- Technical pre-requisites for workshop:
<http://serverless.fi/docs/workshop-preps/>
- This presentation:
<http://serverless.fi/docs/messenger-workshop.pdf>

INTRODUCTION TO MESSENGER BOT COMPONENTS

MESSENGER BOT ARCHITECTURE



MESSENGER PLATFORM

Messenger Platform Beta Launched April 2016

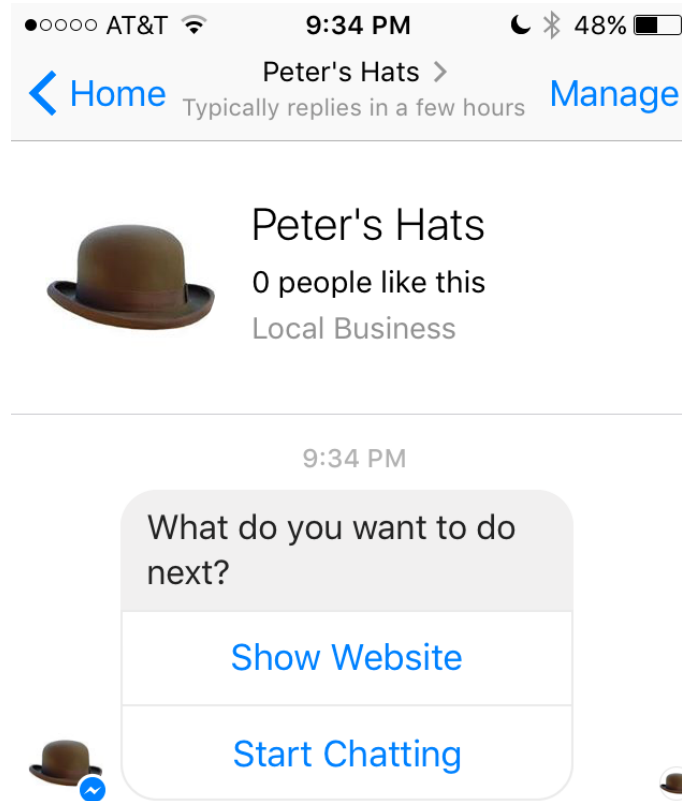
Over 1Bn monthly users

Over 33k bots

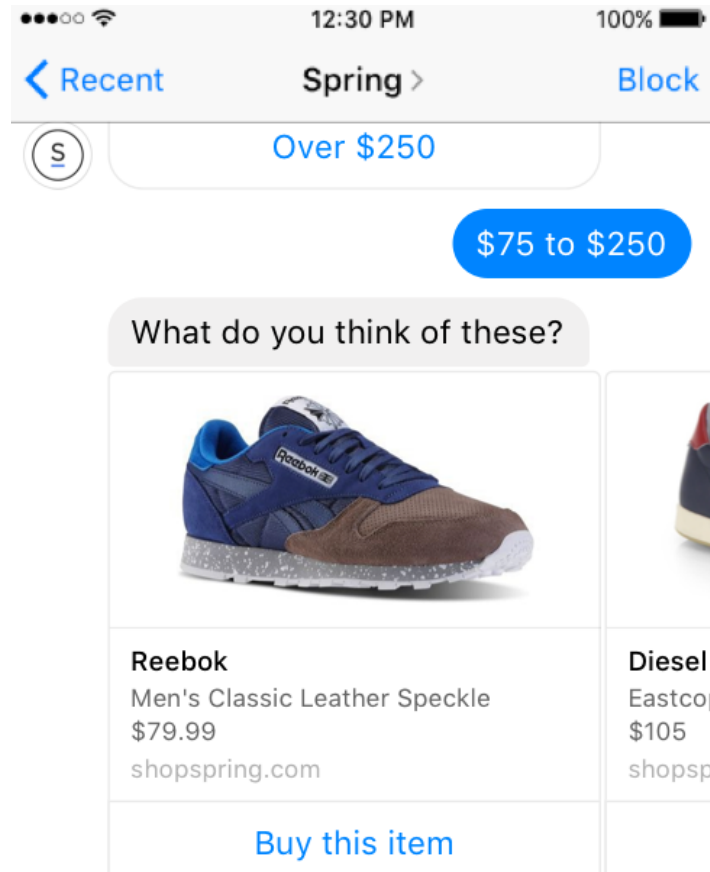
RICH UI ELEMENTS: TEXT



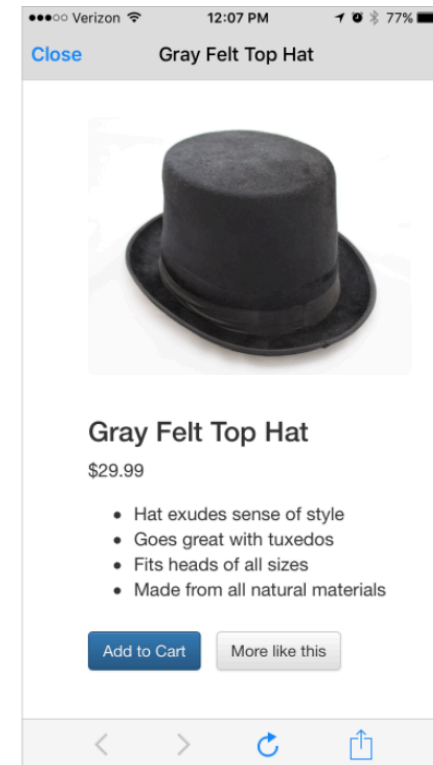
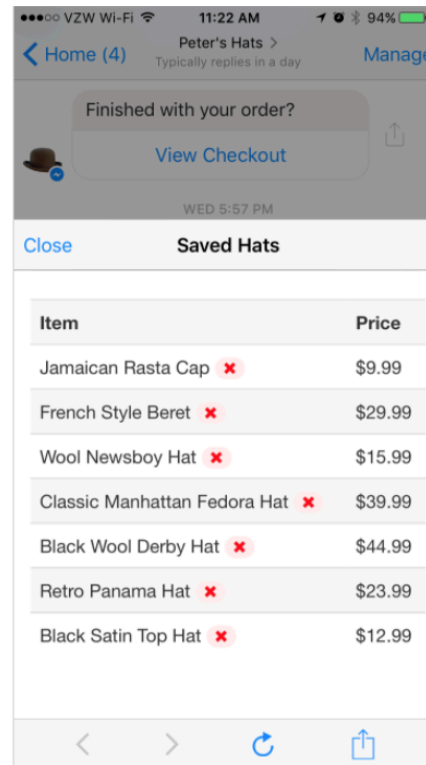
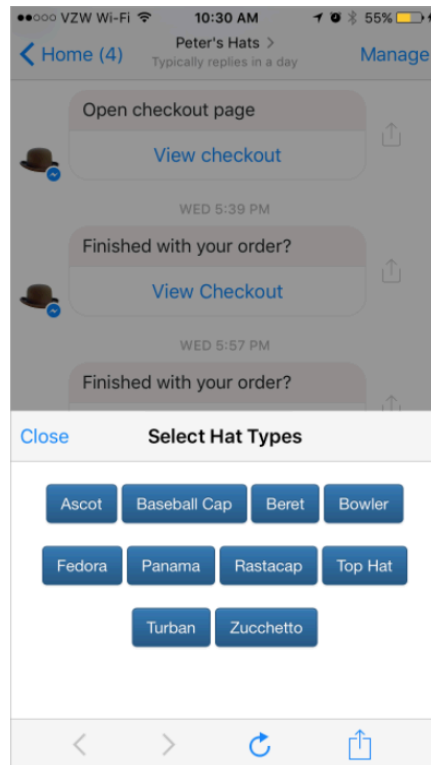
RICH UI ELEMENTS: BUTTONS



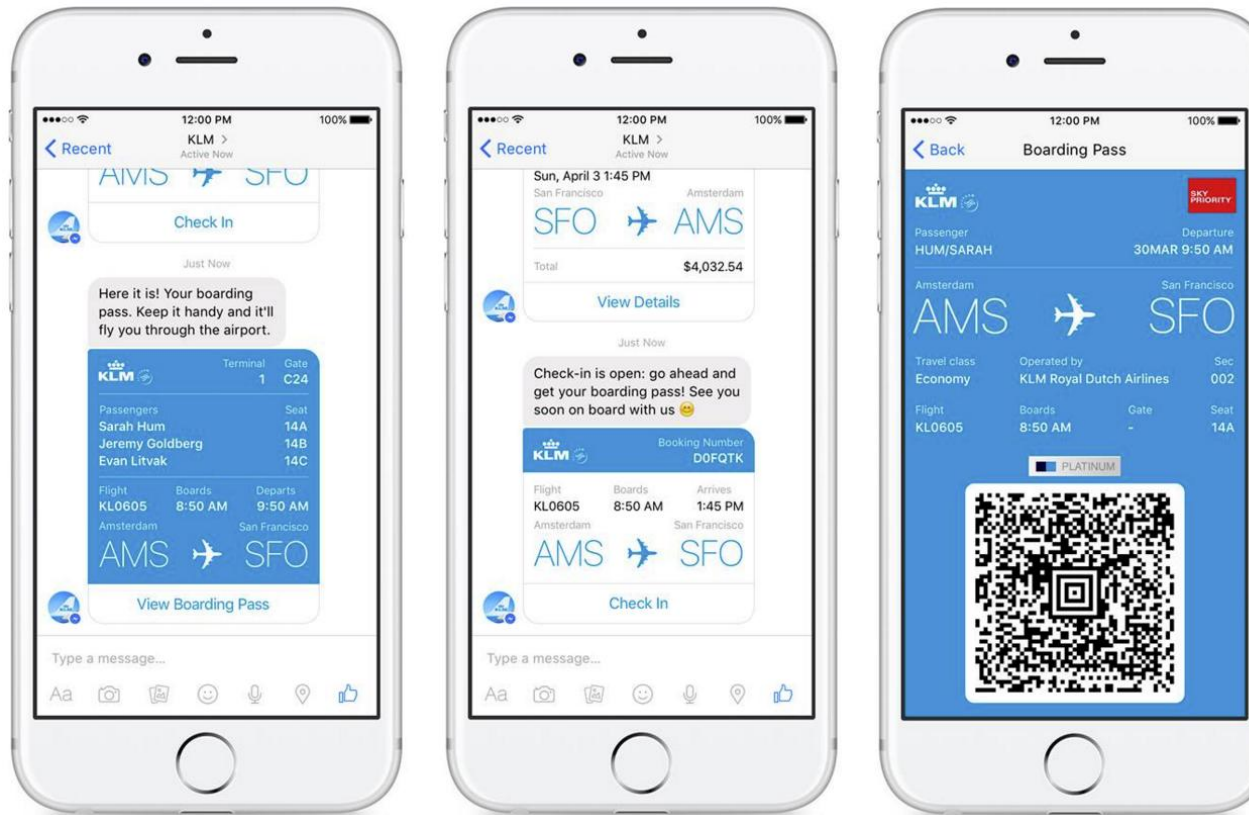
RICH UI ELEMENTS: CAROUSEL



RICH UI ELEMENTS: WEB VIEWS



EXAMPLE: KLM MESSENGER



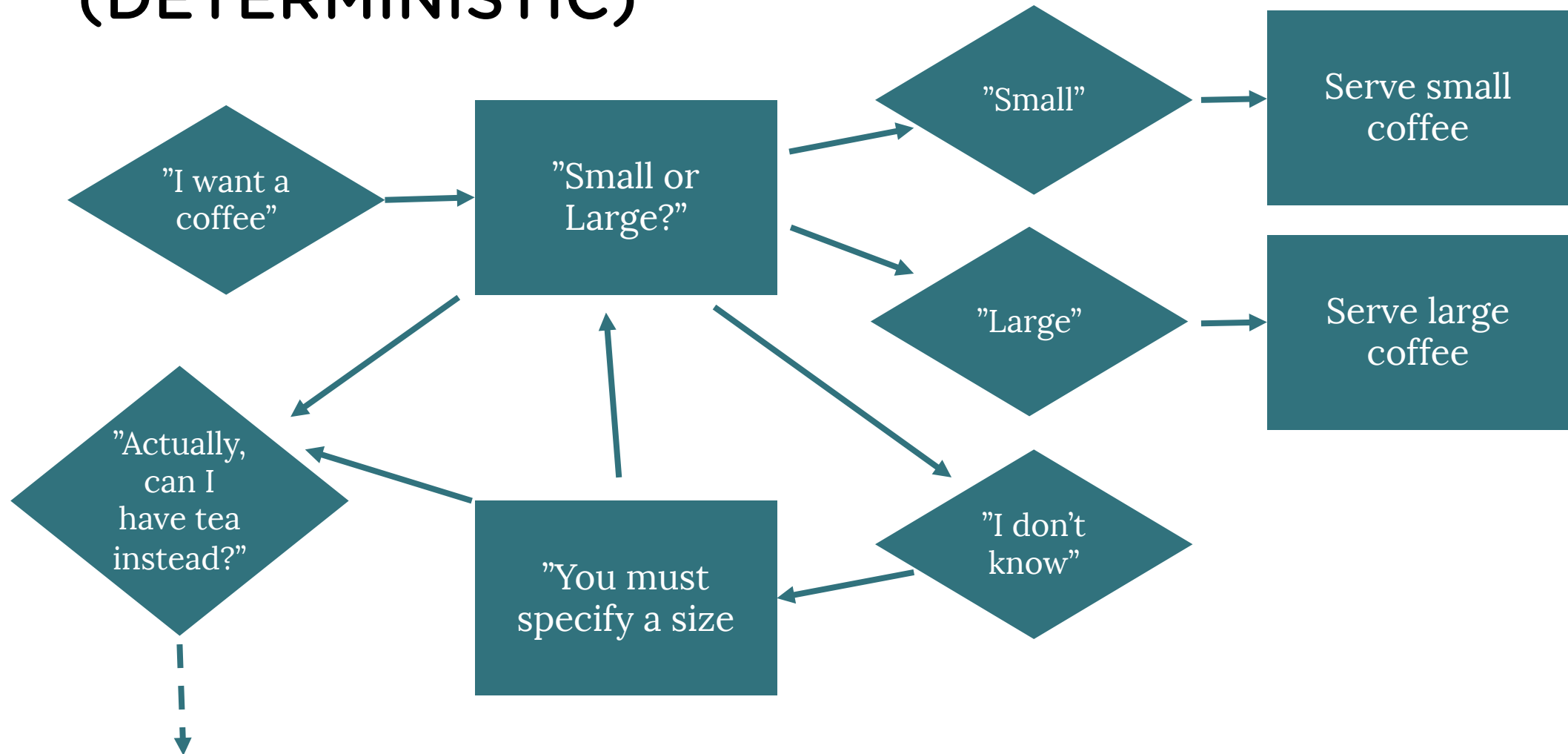
MESSENGER BOT SETUP QUICK GUIDE

1. Create a Facebook page
2. Register a Facebook application
3. Create an endpoint for your bot and hook it to the Facebook app
4. Listen to messages from Messenger
5. Post back responses to the Messenger Platform
6. Get your Facebook application approved to reach the public

Messenger Platform API documentation available at:
<https://developers.facebook.com/docs/messenger-platform/>

WIT.AI

CONVERSATION WORKFLOW (DETERMINISTIC)

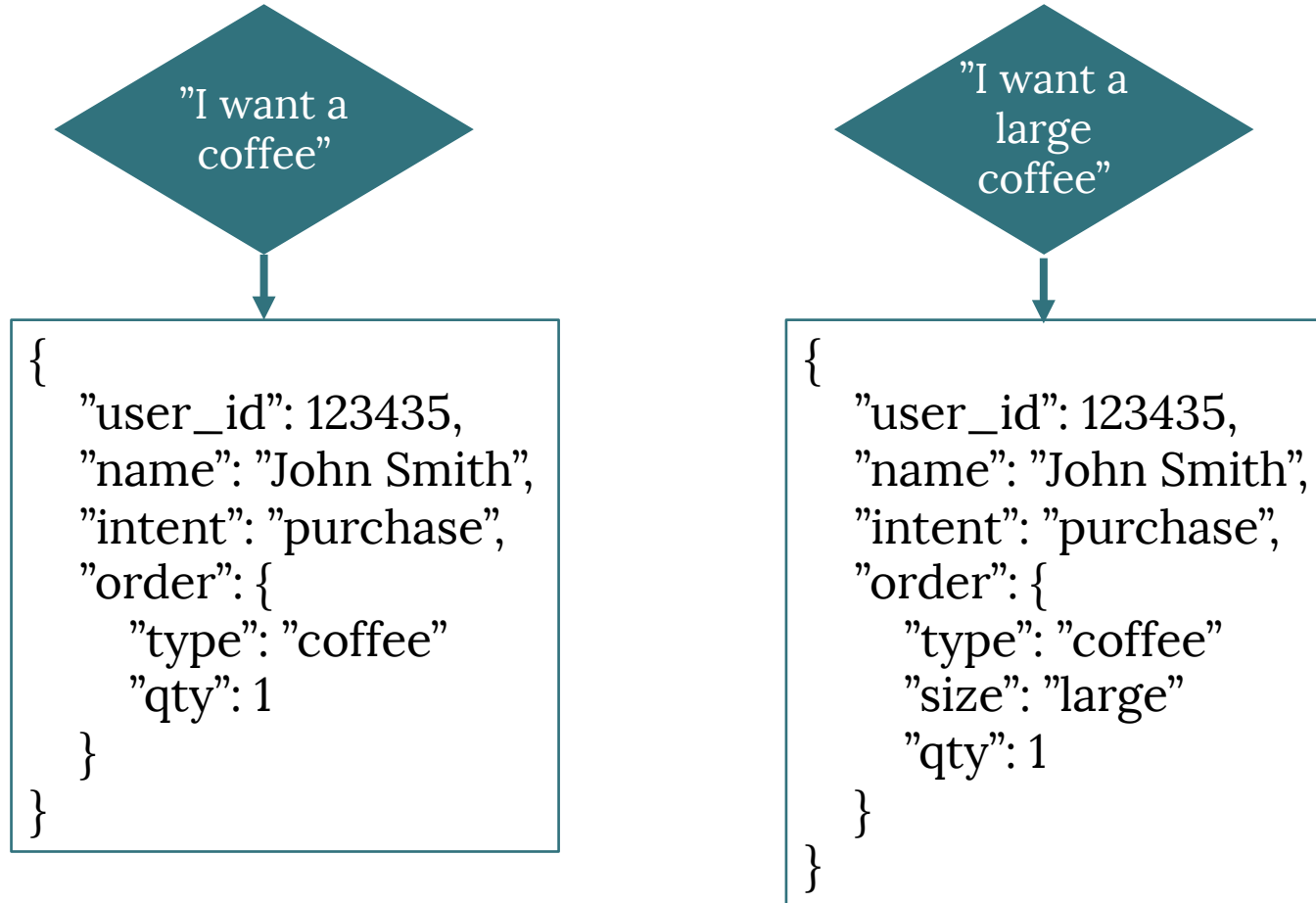


WIT.AI APPROACH

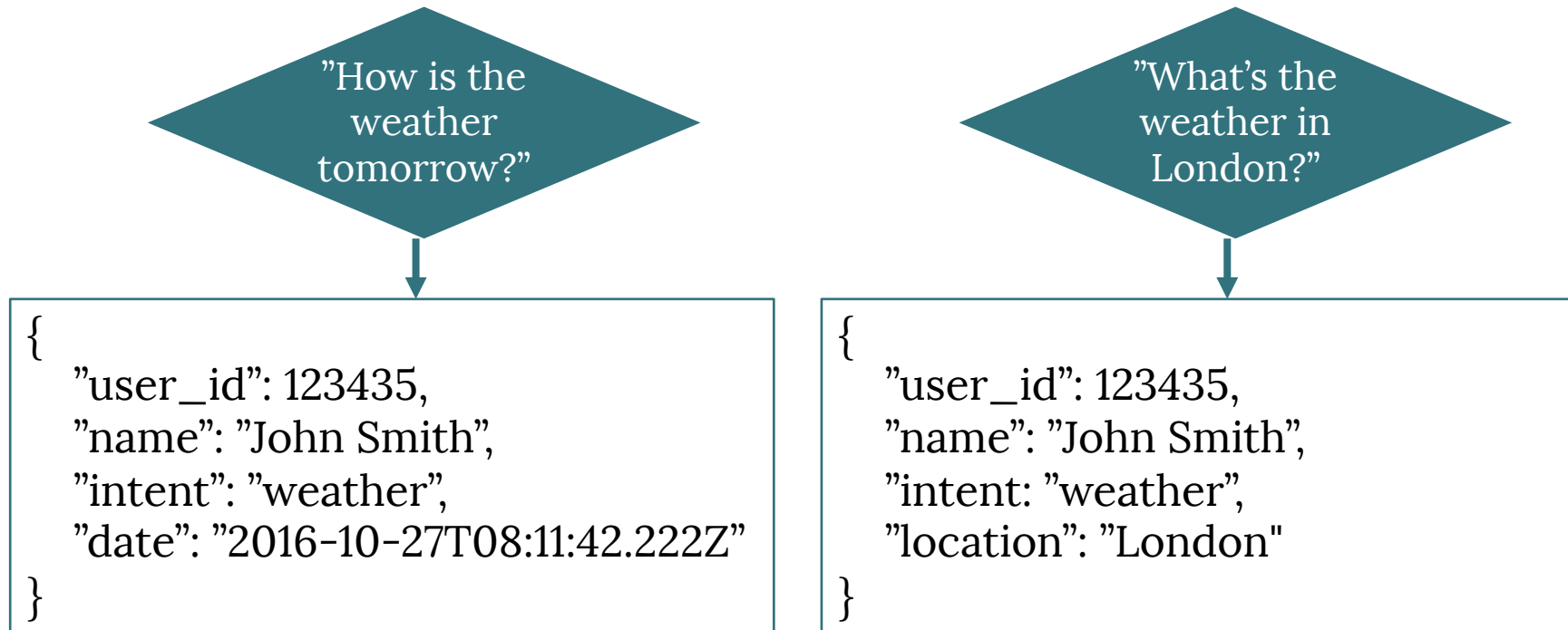
- Stories to define "rules"/"heuristics" for the conversation
- Context to define current state

=> No need for hard coded flows. A combination of stories with rules based on context allow to define complex conversations

CONTEXTS



NATURAL LANGUAGE PROCESSING



SAMPLE WIT.AI CONVERSATION (1)

REQUEST

What's the weather in Brussels?

RESPONSE

```
{
  "type": "merge",
  "entities": {
    "location": [{
      "body": "Brussels",
      "value": {
        "type": "value",
        "value": "Brussels",
        "suggested": true},
      "start": 11,
      "end": 19,
      "entity": "location"}
    ]
  },
  "confidence": 1
}
```

SAMPLE WIT.AI CONVERSATION (2)

REQUEST

```
{  
  "loc": "Brussels"  
}
```

RESPONSE

```
{  
  "type": "action",  
  "action": "fetch-forecast"  
}
```

SAMPLE WIT.AI CONVERSATION (3)

REQUEST

```
{  
  "loc": "Brussels",  
  "forecast": "Sunny"  
}
```

RESPONSE

```
{  
  "type": "msg",  
  "msg": "It's gonna be sunny in Brussels"  
}
```

The *node-wit* module provides higher level method *runActions()* that hides the logic of iterating the contexts with Wit.ai.

Wit.ai HTTP API documentation available at:

<https://wit.ai/docs/http>

Node.js SDK available at

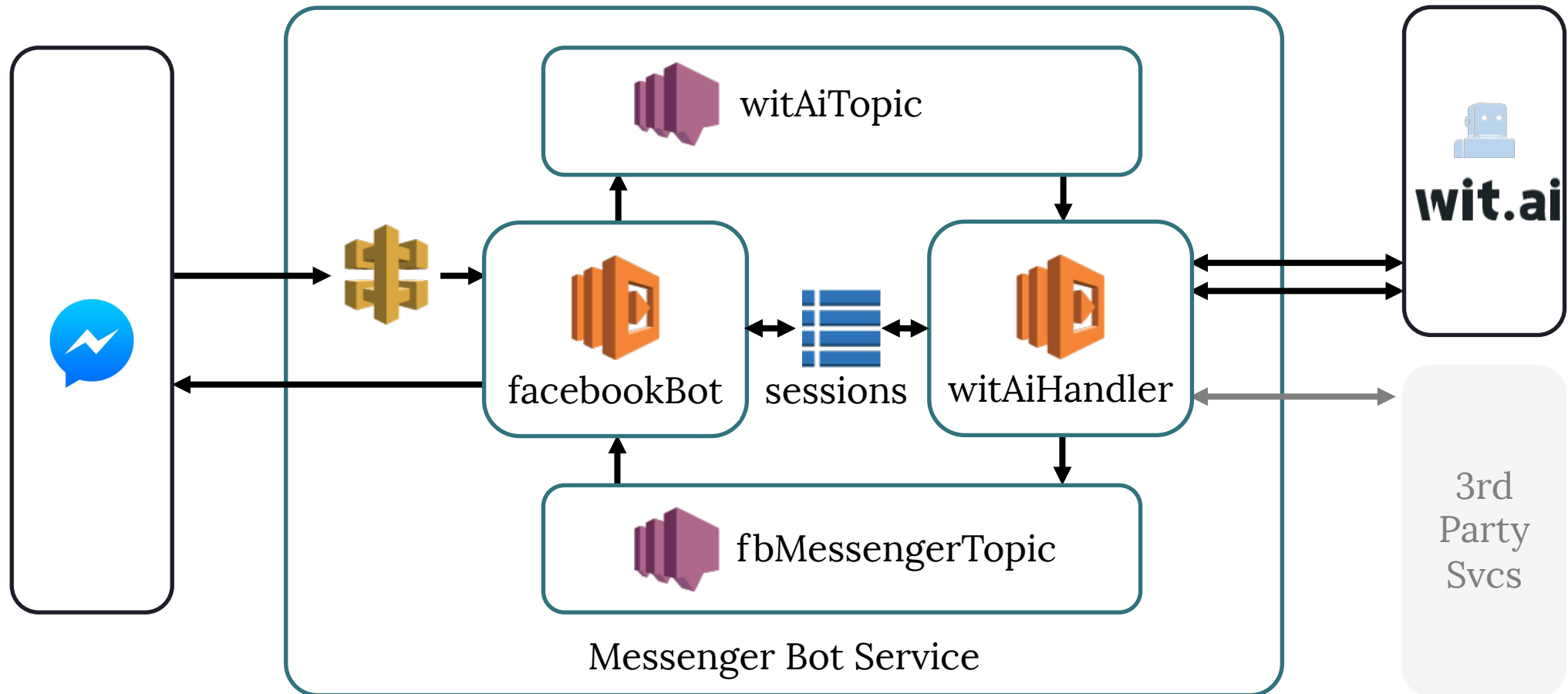
<https://github.com/wit-ai/node-wit>

OUR BOILERPLATE

serverless-messenger-boilerplate and its documentation
available at

<https://github.com/SC5/serverless-messenger-boilerplate>

BOILERPLATE ARCHITECTURE



BOILERPLATE STRUCTURE

| | |
|--------------------------------|--|
| <code>example.env</code> | Template for .env (Facebook / Wit.ai secrets) <= COPY TO .env |
| <code>facebook-bot/</code> | facebookBot function |
| <code>fb-messenger.js</code> | Facebook Messenger interop |
| <code>handler.js</code> | Function entrypoint |
| <code>lib/</code> | Common libraries used by both functions |
| <code>messageQueue.js</code> | Library for reading / publishing to SNS |
| <code>session.js</code> | Library for reading / updating user session (DynamoDB) |
| <code>package.json</code> | ADD NODE MODULES REQUIRED BY YOUR BOT HERE |
| <code>serverless.yml</code> | Service configuration (functions, endpoints, resources) |
| <code>test/</code> | Serverless-mocha-plugin tests |
| <code>facebookBot.js</code> | Tests for facebookBot function |
| <code>witAiHandler.js</code> | Tests for witAiHandler function |
| <code>webpack.config</code> | Configuration for Webpack deployment (serverless-webpack plugin) |
| <code>wit-ai/</code> | witAiHandler function |
| <code>handler.js</code> | Entrypoint for function |
| <code>my-wit-actions.js</code> | Your bot logic (Wit.ai actions) <= IMPLEMENT YOUR BOT LOGIC HERE |
| <code>wit-ai.js</code> | Bot logic built interop on Wit.ai |

.ENV FILE (FROM EXAMPLE.ENV)

| | |
|--------------------------------|---|
| FACEBOOK_BOT_VERIFY_TOKEN | User defined verification token for the Facebook App |
| FACEBOOK_BOT_PAGE_ACCESS_TOKEN | Facebook-generated access token for the page |
| WIT_AI_TOKEN | API token for Wit.ai |
| FACEBOOK_ID_FOR_TESTS | Facebook ID used to post messages in tests. Available from the sessions table |
| SERVERLESS_PROJECT | Service name (keep in sync with service name in serverless.yml) |

DECOUPLING WITH SNS

- Lambda SNS subscriptions: see [serverless.yml](#)
- Posting and decoding messages done with [messageQueue.js](#)
- Sample endpoint at [facebookBot/handler.js](#)
- In production environments, SNS could be strengthened e.g. with SQS to guarantee delivery

LOCAL DEVELOPMENT AND TESTING (SERVERLESS-MOCHA-PLUGIN)

- serverless-mocha-plugin included in boilerplate with predefined tests for facebookBot and witAiHandler functions in the test/ directory
- Run all tests : *sls invoke test*
- Run tests for facebookBot : *sls invoke test -f facebookBot*
- Run tests for witAiHandler: *sls invoke test -f witAiHandler*
- (Create new tests with *sls create test -f functionName*)
- Comment out line that sets *process.env.SILENT* in *test/*.js* if you want messages to be sent during testing

OPTIMIZED DEPLOYMENT (SERVERLESS-WEBPACK)

- serverless-webpack plugin included to streamline deployment package and accelerate cold start of Lambda functions
- Pre-configured in webpack.config

BUILDING A WEATHER BOT

SETUP PROJECT

1. CREATE SERVERLESS PROJECT

```
> sls install -u https://github.com/SC5/serverless-messenger-boilerplate
```

```
> mv serverless-messenger-boilerplate weather-bot
```

- Change service name to e.g. “*weather-bot*” in *serverless.yml*, then install node modules and copy *example.env* to *.env*

```
> npm install
```

```
> cp example.env .env
```

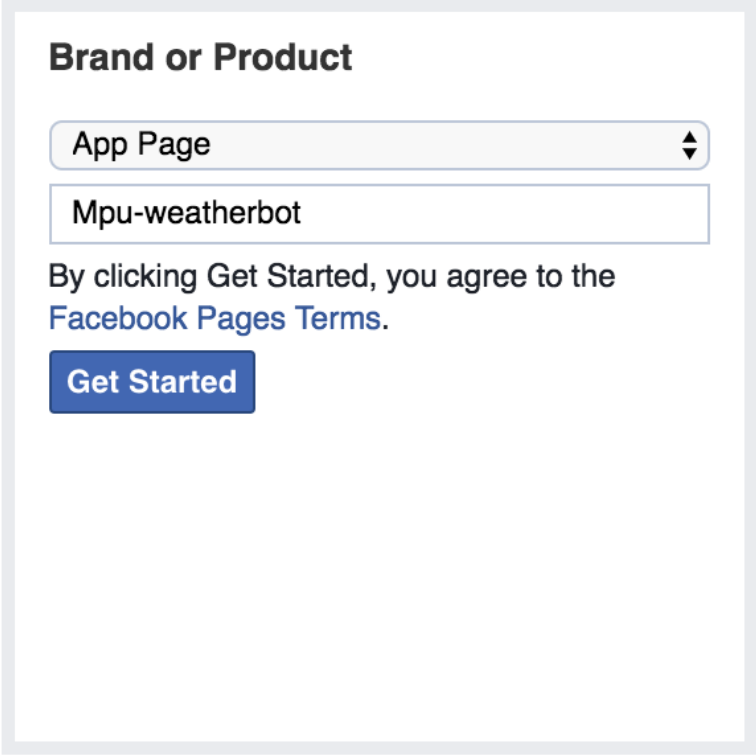
- Generate a random verification token to *VERIFY_TOKEN* in *.env*
- Deploy the service and memorize the URL

```
> sls deploy
```

SETUP FACEBOOK PAGE AND APPLICATION

CREATE A FACEBOOK PAGE

- Go to <http://facebook.com/pages/create>
- Create new page
- Skip all following steps



Brand or Product

App Page

Mpu-weatherbot

By clicking Get Started, you agree to the [Facebook Pages Terms](#).

Get Started

CREATE FACEBOOK APPLICATION

- Go to <https://developers.facebook.com/quickstarts/?platform=web>
- Click "Skip and Create App ID"
- Fill in info
- Create app ID

Create a New App ID

Get started integrating Facebook into your app or website

Display Name

mpu-weatherbot

Contact Email

mikael.puittinen@sc5.io

Category

Apps for Pages ▼

By proceeding, you agree to the [Facebook Platform Policies](#)

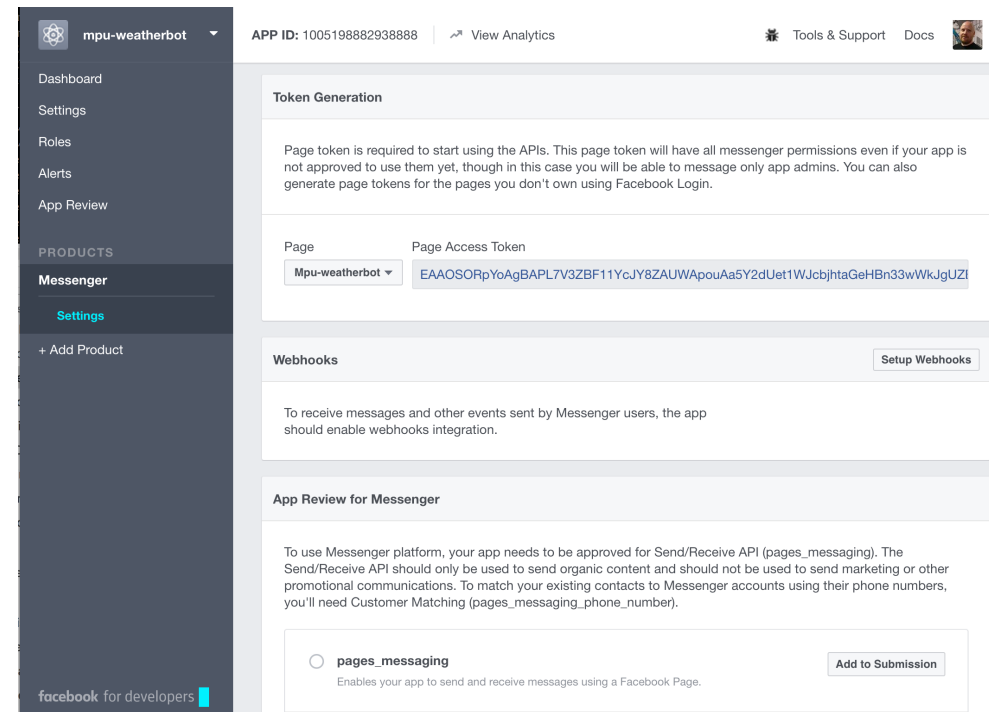
Cancel

Create App ID

SETUP MESSENGER TOKEN

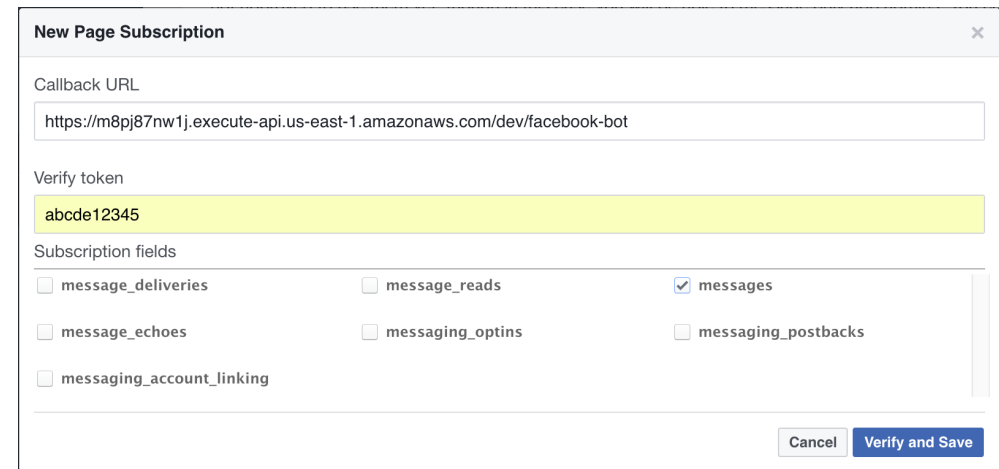
- "Get Started" for Messenger
- Generate a token for your page
- Copy token to `.env`
(`FACEBOOK_BOT_PAGE_ACCESS_TOKEN`)
- Save changes and deploy your service

```
> sls deploy
```



SETUP MESSENGER WEBHOOK

- Initiate "Setup Webhooks"
- Enter the endpoint URL that you got during deployment
- Enter your verify token from *.env* (FACEBOOK_VERIFY_TOKEN)
- Verify and Save

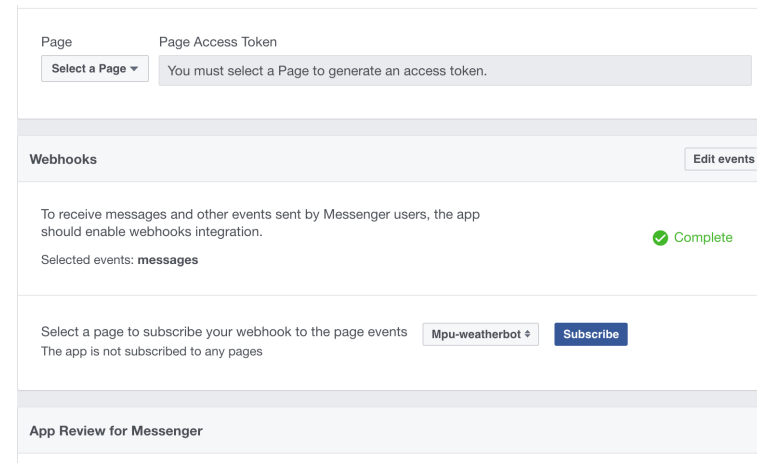


The screenshot shows the 'New Page Subscription' dialog box. It contains the following fields and options:

- Callback URL:** A text input field containing the URL `https://m8pj87nw1j.execute-api.us-east-1.amazonaws.com/dev/facebook-bot`.
- Verify token:** A text input field containing the token `abcde12345`, which is highlighted in yellow.
- Subscription fields:** A list of checkboxes for selecting the types of events to receive:
 - ☐ message_deliveries
 - ☐ message_echoes
 - ☐ message_reads
 - ☐ messaging_account_linking
 - ☐ messaging_optins
 - ☒ messages
 - ☐ messaging_postbacks
- Buttons:** At the bottom right, there are two buttons: 'Cancel' and 'Verify and Save'.

SUBSCRIBE THE WEBHOOK TO THE FACEBOOK PAGE

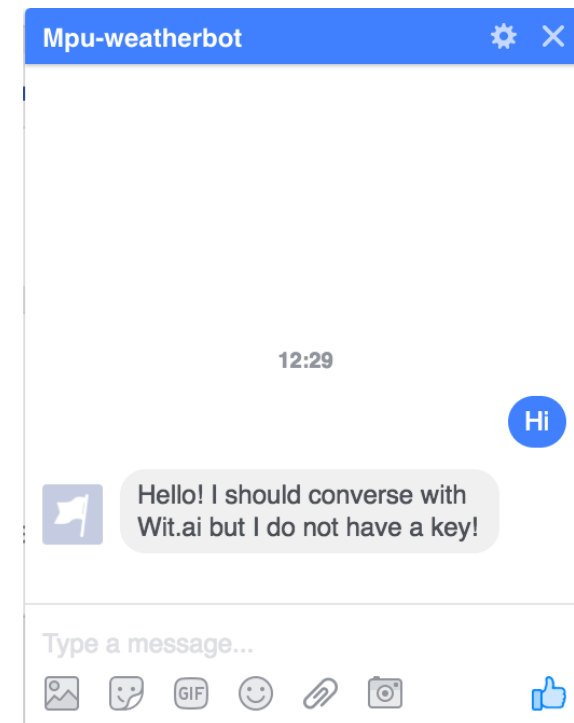
1. Under "Webhooks", select your page and subscribe



The screenshot shows the Facebook Developer console interface for configuring webhooks. At the top, there are two tabs: "Page" and "Page Access Token". Under the "Page" tab, there is a dropdown menu labeled "Select a Page" and a message: "You must select a Page to generate an access token." Below this, the "Webhooks" section is active, showing a status of "Complete" with a green checkmark. It indicates that the app should enable webhooks integration and that the selected events are "messages". At the bottom of the "Webhooks" section, there is a prompt to "Select a page to subscribe your webhook to the page events" with a dropdown menu showing "Mpu-weatherbot" and a "Subscribe" button. Below the "Webhooks" section, there is a section for "App Review for Messenger".

TRY IT

- Send message to your Facebook page



FACEBOOK APPLICATION ACCESS

Access to unapproved applications is limited to developers only.

If you want to give access to other users, you should provide "Developer" / "Testers" roles to them from the "Roles" panel (accessible from the left sidebar)

SET UP OPENWEATHER API

REGISTER TO OPENWEATHERMAP

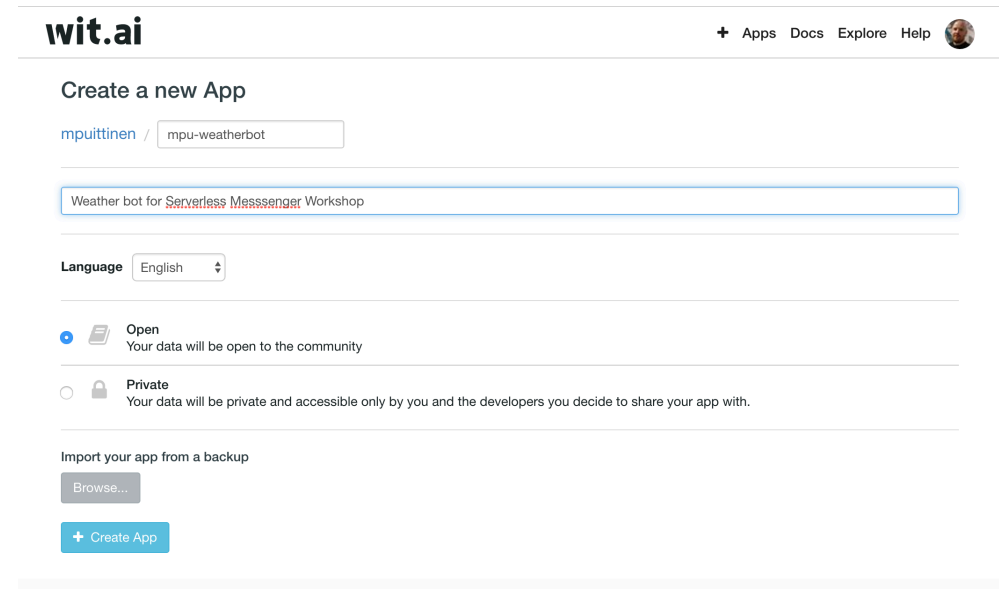
1. Go to <https://openweathermap.org/appid>
2. Sign up
3. Copy API key and set it to `WEATHER_API_TOKEN` in `.env`

| Key | Name |
|----------------------------------|---------|
| 83ffbd8ffb5bd3cded626ffddfbd67f6 | Default |

SETUP WIT.AI

REGISTER + CREATE WIT.AI APP

- Go to <https://wit.ai>
- Register (if not already done)
- Create a new App



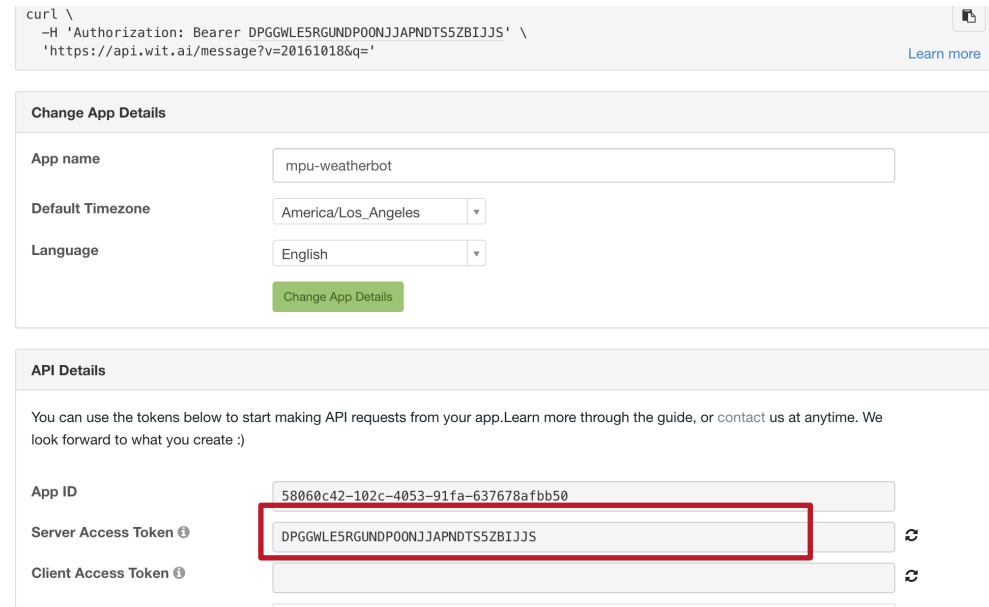
The screenshot shows the 'Create a new App' page on the wit.ai website. The header includes the 'wit.ai' logo and navigation links for '+ Apps', 'Docs', 'Explore', 'Help', and a user profile icon. The main form area is titled 'Create a new App' and contains the following elements:

- A username field with 'mpuittinen' and a slash, followed by an app name input field containing 'mpu-weatherbot'.
- A description input field containing 'Weather bot for ~~Serverless~~ Messenger Workshop'.
- A 'Language' dropdown menu set to 'English'.
- Two radio button options for data privacy:
 - Open** (selected): 'Your data will be open to the community'.
 - Private**: 'Your data will be private and accessible only by you and the developers you decide to share your app with.'
- An 'Import your app from a backup' section with a 'Browse...' button.
- A blue '+ Create App' button at the bottom.

CONNECT THE ENDPOINT WITH WIT.AI

- Go to Settings
- Copy the Server Access Token ID to *.env* (*WIT_AI_TOKEN*)
- Deploy your service

```
> sls deploy
```



The screenshot shows the Wit.ai 'Change App Details' and 'API Details' sections. The 'API Details' section contains the App ID, Server Access Token, and Client Access Token. The Server Access Token is highlighted with a red box.

```
curl \
-H 'Authorization: Bearer DPGGWLE5RGUNDPO0NJJAPNDTS5SZBIJJS' \
'https://api.wit.ai/message?v=20161018&q='
```

Change App Details

App name: mpu-weatherbot

Default Timezone: America/Los_Angeles

Language: English

[Change App Details](#)

API Details

You can use the tokens below to start making API requests from your app. Learn more through the guide, or contact us at anytime. We look forward to what you create :)

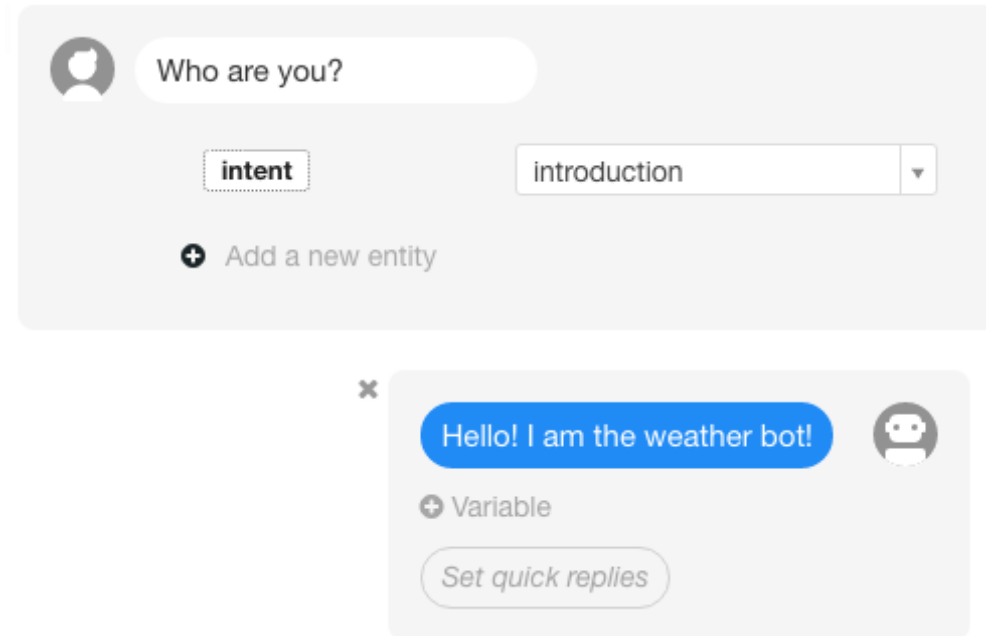
App ID: 58060c42-102c-4053-91fa-637678afbb50

Server Access Token ⓘ: DPGGWLE5RGUNDPO0NJJAPNDTS5SZBIJJS

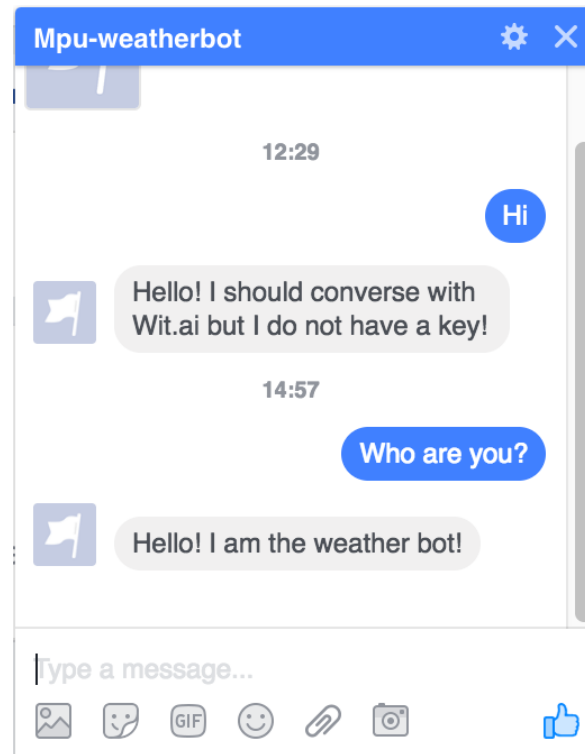
Client Access Token ⓘ:

LET THE BOT INTRODUCE ITSELF

- Create a new story
- Add user input and intent
- Add a response with "Bot Sends"
- Remember to save the story



TRY IT!



ADD LOGIC TO THE STORY

- Add a new story
- Type in your user input
- Add entity *intent* with value *weather*
- Select the location and add new entity *wit/location*
- Select the date and add new entity *wit/datetime*
- Add action *getWeather* with "Bot Executes"
- Set context key *location && datetime && description && temperature*
- Add a response using "Bot sends"
- Save

The screenshot displays the Wit.ai interface for creating a new story. At the top, a user input box contains the text "What is the weather in Helsinki tomorrow?". Below this, two entities are defined: "wit/datetime" with the value "19/10/2016, 00:00:00" and "wit/location" with the value "Helsinki". A dropdown menu for "intent" is set to "weather". Below the entities, there is a button to "Add a new entity". The action section shows the function "getWeather (context, entities)" with a lightning bolt icon. The context keys are listed as "location && datetime && description && temperature". The response section shows a blue bubble with the text "The weather in {location} {datetime} is {description} with temperature of {temperature}°C". Below the response, there is a "Variable" section with a "Set quick replies" button.

IMPLEMENT LOGIC FOR FETCHING WEATHER

- Copy snippet `getWeather.js` from `examples/weather-bot` to `my-wit-actions.js`
- Copy file `weather.js` from `examples/weather-bot` to the `wit-ai` directory

```
> sls deploy function -f witAiHandler
```

```
1  const moment = require('moment');
2  const weather = require('./weather');
3
4  const actions = {
5    getWeather: (data) => new Promise((resolve, reject) => {
6      const context = data.context;
7      const entities = data.entities;
8
9      const missingLocation = entities.location === undefined;
10     const location = entities.location ? entities.location[0].value : null;
11     const datetime = entities.datetime ? entities.datetime[0].value : null;
12
13     if (missingLocation) {
14       const contextData = Object.assign({}, context, { missingLocation });
15       resolve(contextData);
16     } else if (datetime) {
17       weather.forecastByLocationName(location, datetime)
18         .then((weatherData) => {
19           const contextData = Object.assign({}, context, weatherData);
20           if (datetime) {
21             Object.assign(contextData, { datetime: moment(datetime).calendar().toLowerCase() });
22           }
23           resolve(contextData);
24         })
25         .catch(reject);
26     } else {
27       weather.weatherByLocationName(location)
28         .then((weatherData) => {
29           const contextData = Object.assign({}, context, weatherData);
30           if (datetime) {
31             Object.assign(contextData, { datetime: moment(datetime).calendar().toLowerCase() });
32           }
33           resolve(contextData);
34         })
35         .catch(reject);
36     }
37   });
38 }
```

TRY IT!

- An inquiry with date and location information works!
- But if we drop out the date, we do not get the right response.

Let's fix that!



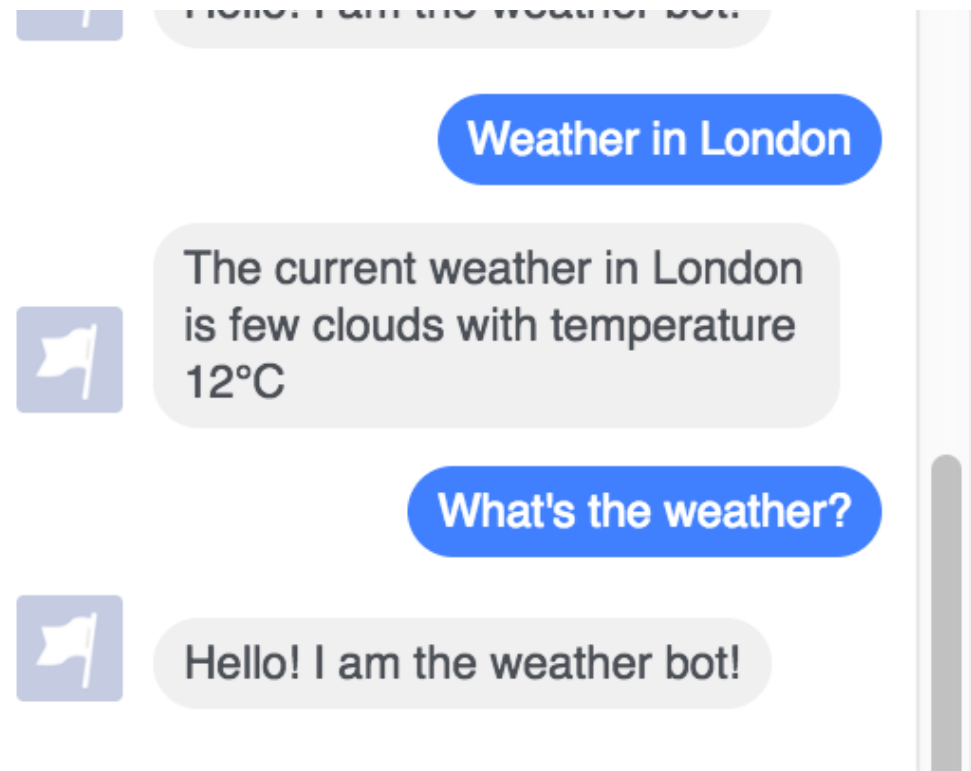
FORK THE STORY WITHOUT A DATE

- Click the fork icon next to the context keys
- Create new context key *location && description && temperature* (without *datetime*)
- Create new response
- Save

The screenshot shows a story editor interface. At the top, a light gray box contains the text `getWeather (context, entities)` next to a lightning bolt icon. Below this, two context key boxes are shown: the left one contains `location && datetime && description && temperature` and the right one contains `location && description && temperature` in blue text. A fork icon (two arrows pointing away from each other) is to the right of the second box. Below these, a blue rounded rectangle contains the text: "The current weather in {location} is {description} with temperature {temperature}°C". To the right of this box is a robot icon. Below the blue box, there is a "+ Variable" label and a button labeled "Set quick replies".

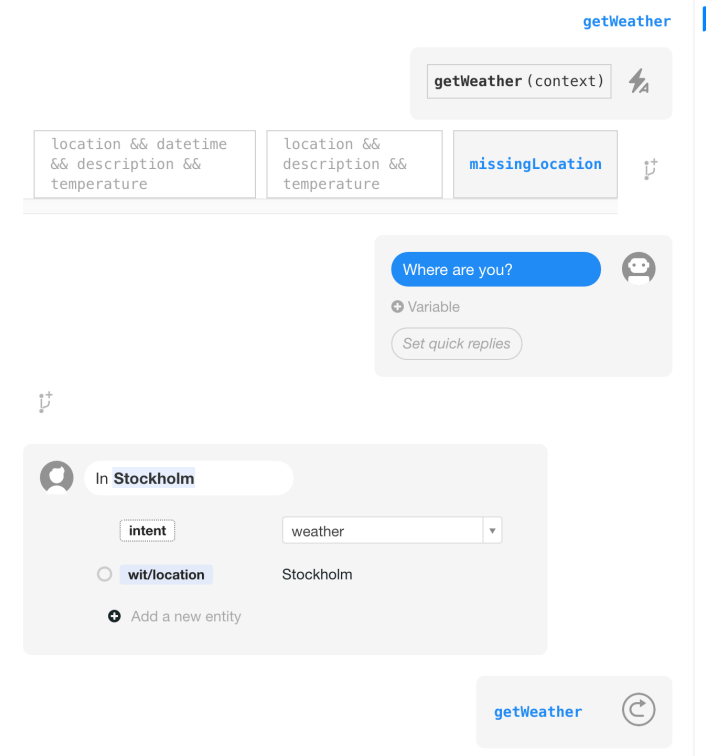
TRY IT!

- The current weather is now given if no date is provided
- But if we do not provide a location, the bot should probably ask for it!



ASK ADDITIONAL QUESTIONS FROM THE USER

- Click the bookmark icon next to the *getWeather* action and create a bookmark *getWeather*
- Create new fork with context key *missingLocation*
- Add a bot response for asking the location
- Add a user response and set *intent* and *wit/location* entities
- Add Jump to *getWeather* bookmark created earlier



TRY IT!

IT WORKS!



TRAINING THE BOT

WIT.AI UNDERSTANDING

- Use Wit.ai understanding to train model
- Enter expressions, set entities and validate to improve accuracy

Test how your bot understands a sentence

You can train your bot by adding more examples

What is the temperature in **Frankfurt on Sunday**?

☐ wit/datetime 23/10/2016, 00:00:00

☐ wit/location Frankfurt


☐ intent weather

[Add a new entity](#)

[Validate](#)

Your app uses 3 entities

| Name | Search Strategy | Values |
|------|-----------------|--------|
|------|-----------------|--------|

Press  to chat with your bot

WIT.AI INBOX

- Use Wit.ai inbox to train model based on incoming messages
- Validate expressions that have been entered by users
- Preview API output for expressions

The screenshot shows the Wit.ai Expressions interface. At the top, there are tabs for 'Expressions', 'Voice', and 'Stories'. The 'Expressions' tab is active. Below the tabs, there is a text input field containing 'Who are you=' and a dropdown menu showing 'introduction'. Below the input field, there is a button labeled 'intent' and a button labeled 'Add a new entity'. A green button labeled 'Validate' is visible. To the right of the 'Validate' button, there is a button labeled 'Preview API Output'. Below the 'Validate' button, there is a large text area displaying the JSON output of the API call:

```
[
  {
    "_text": "Who are you=",
    "confidence": null,
    "intent": "default_intent",
    "entities": {
      "intent": [
        {
          "confidence": 0.9827075899547016,
          "value": "introduction"
        }
      ]
    }
  }
]
```

SC5

THANK YOU!

mikael.puittinen@sc5.io / @mpuittinen

eetu.tuomala@sc5.io / @hopeatussi