SC5

SERVERLESS MESSENGER BOT WORKSHOP

Mikael Puittinen, Chief Technology Officer

Eetu Tuomala, Cloud Application Architect

10.11.2016

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

DNA

elisa

Enegia

Gasum

HAPPYORNOT

KEMPPI

VEIKKAUS

POP Vakuutus

sanoma

TeliaSonera

WÄRTSILÄ

L&T

amazon web services Partner Network

CONSULTING PARTNER

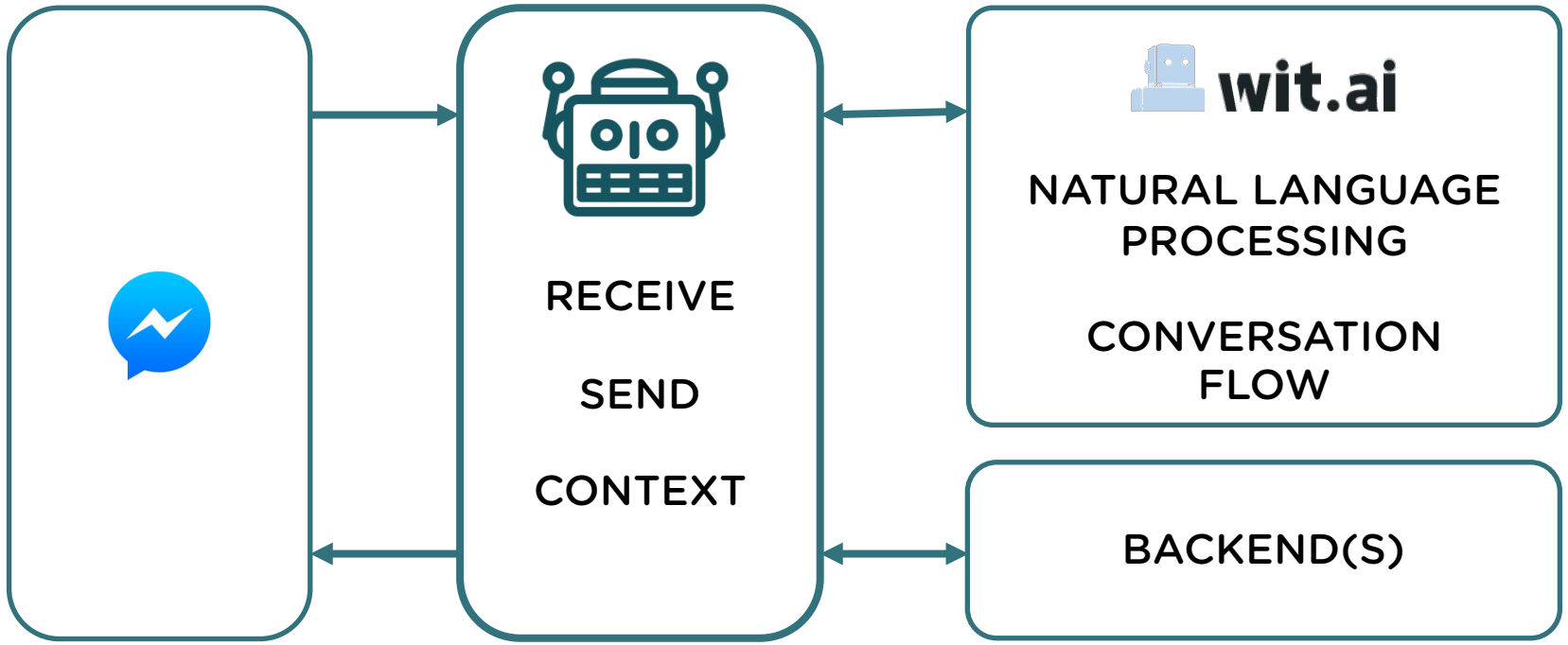
VISIT OUR WEB SITE FOR MORE INFO: HTTPS://SC5.IO

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



SC5

MESSENGER PLATFORM

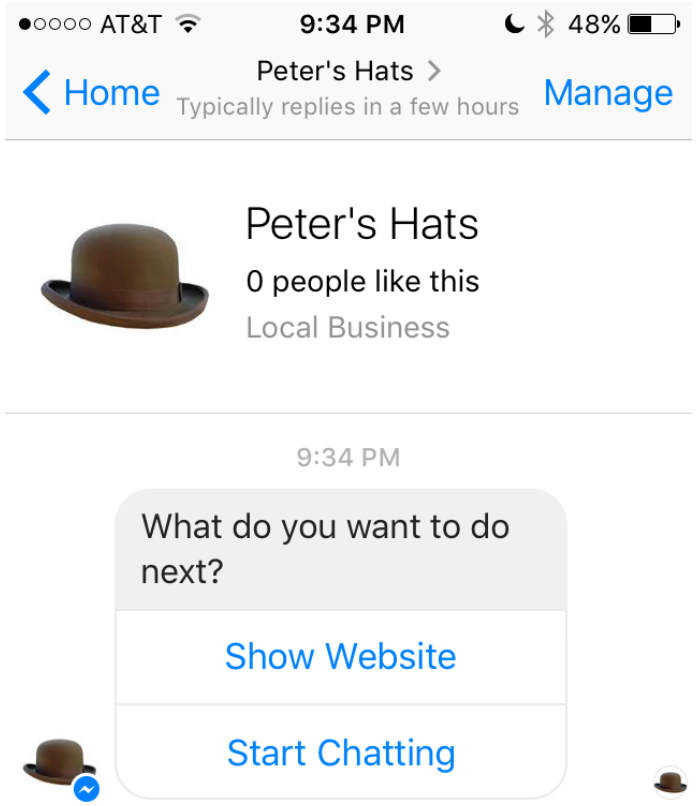
Messenger Platform Beta Launched April 2016

Over 1Bn monthly users

Over 33k bots

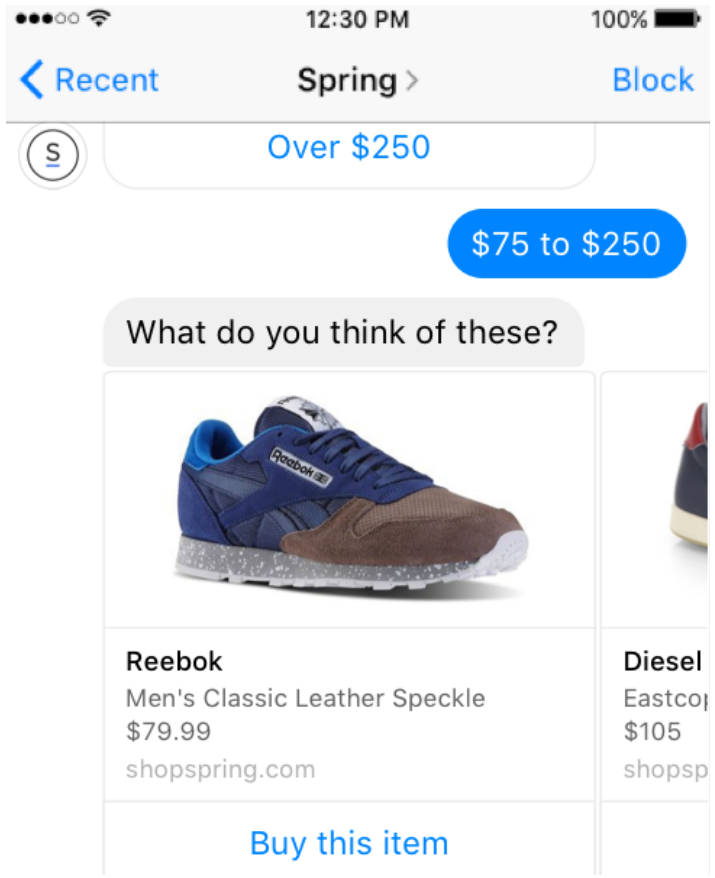
RICH UI ELEMENTS: TEXT

RICH UI ELEMENTS: BUTTONS



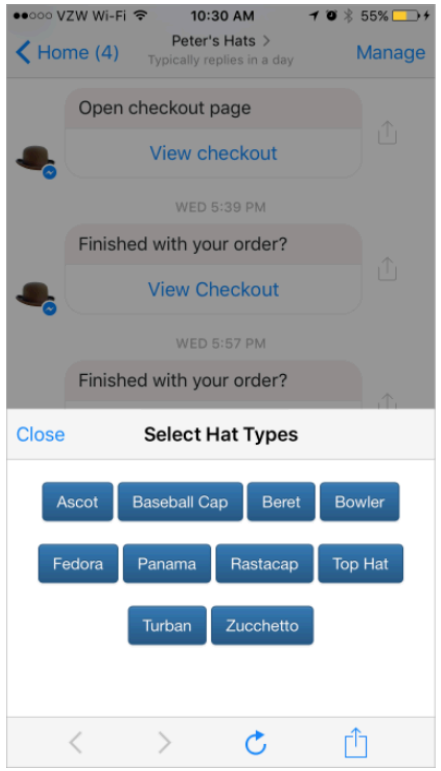
SC5

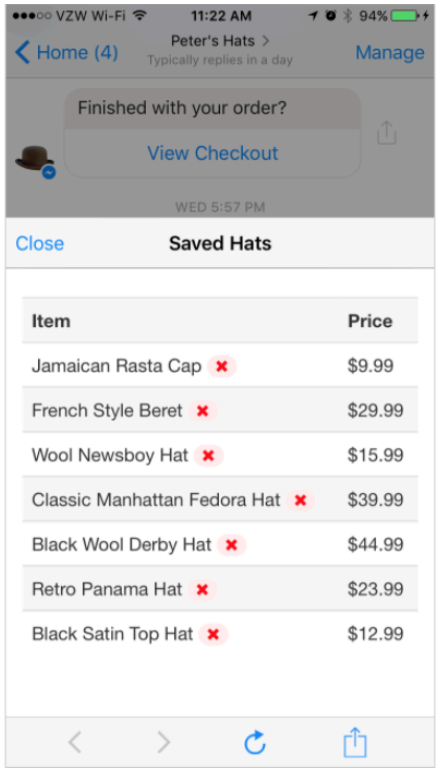
RICH UI ELEMENTS: CAROUSEL

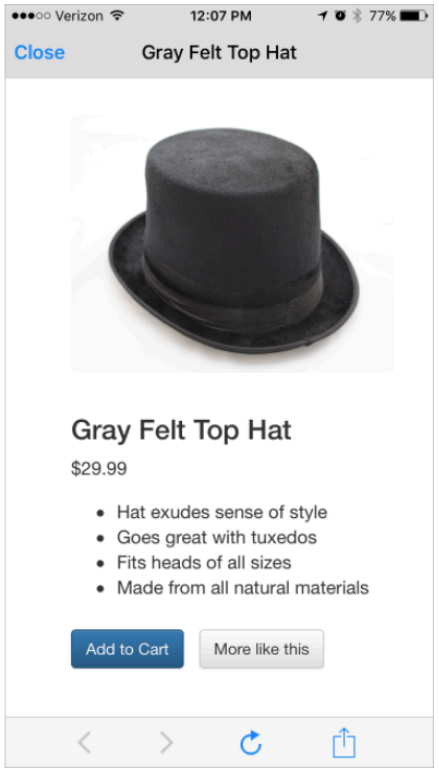


SC5

RICH UI ELEMENTS: WEB VIEWS

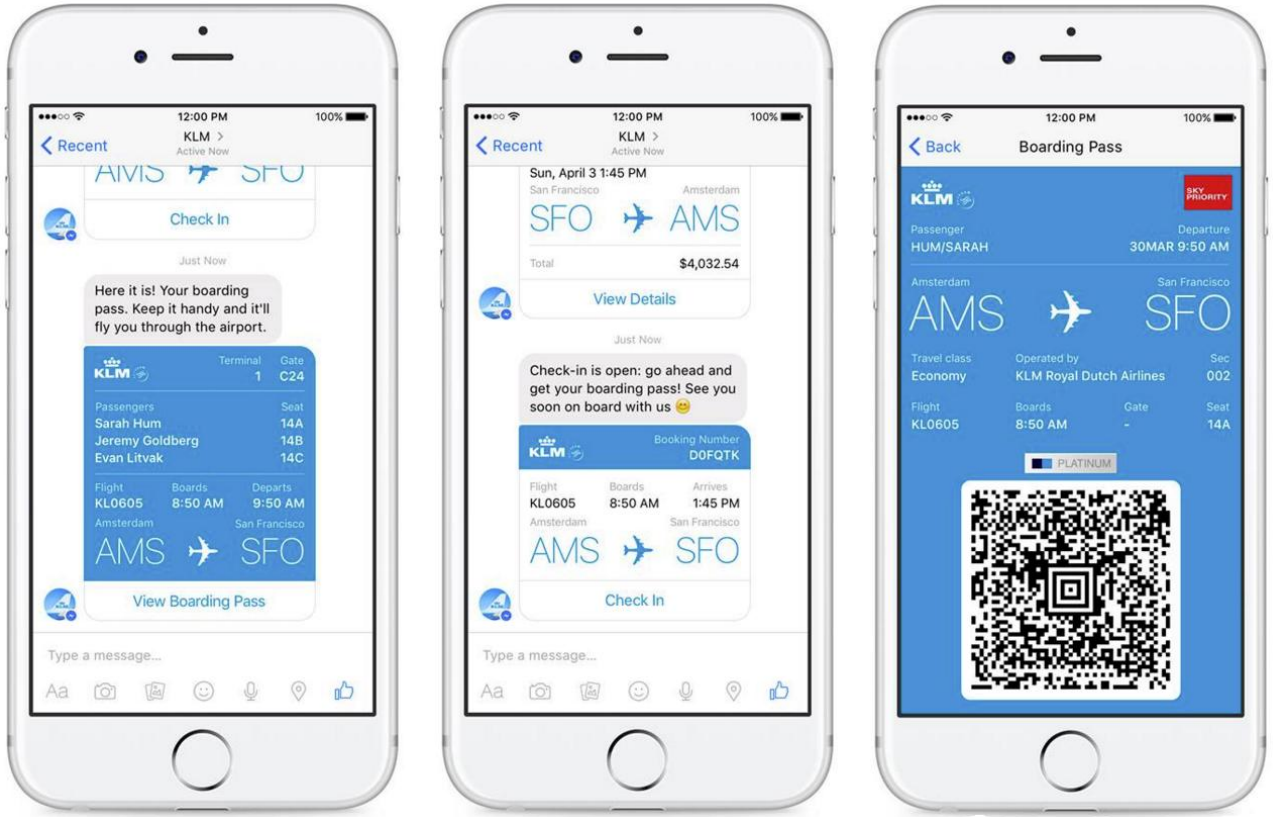






SC5

EXAMPLE: KLM MESSENGER



SC5

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

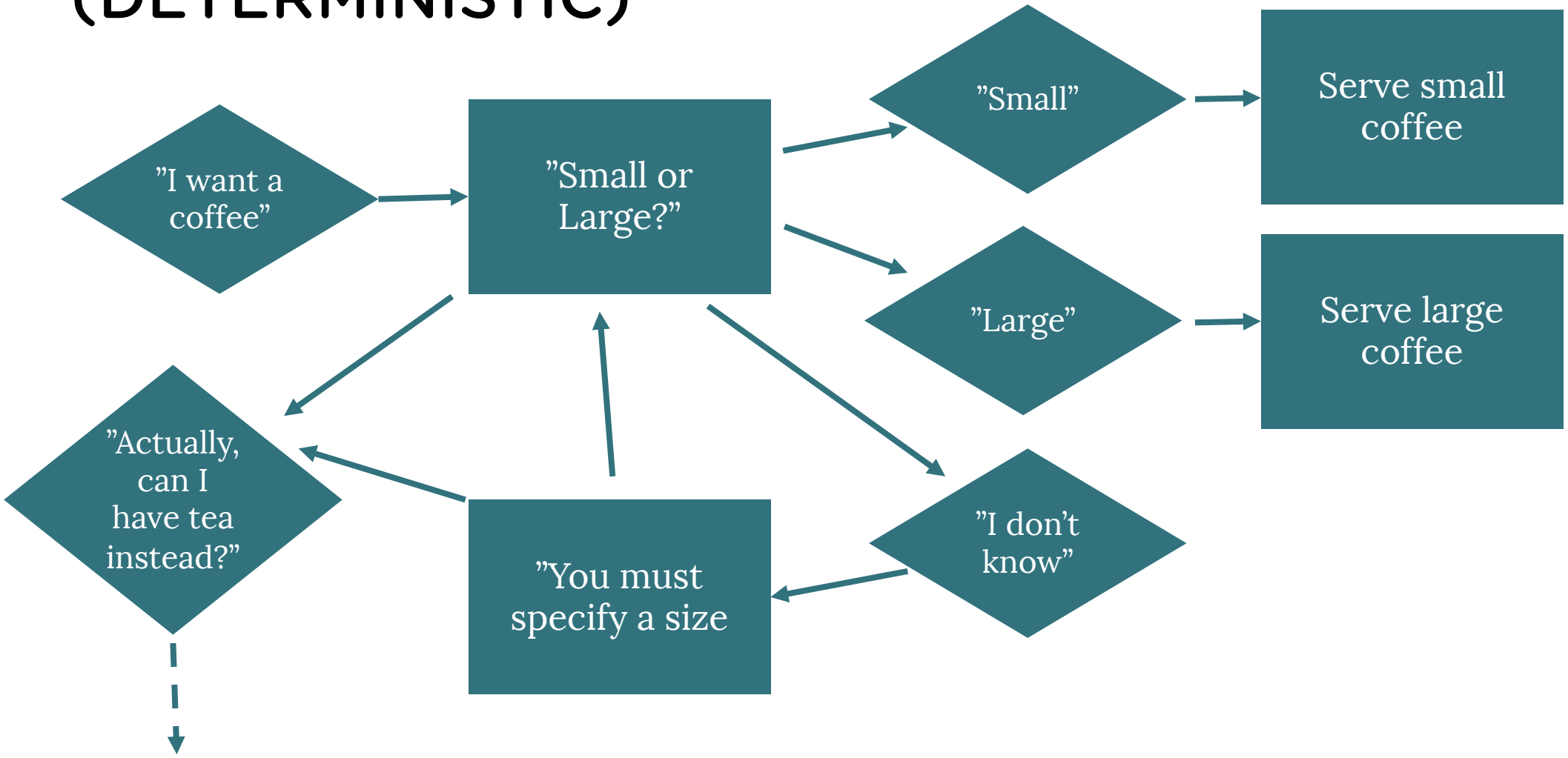
SC5

Messenger Platform API documentation available at:

https://developers.facebook.com/docs/messenger-platform/

WIT.AI

CONVERSATION WORKFLOW (DETERMINISTIC)



SC5

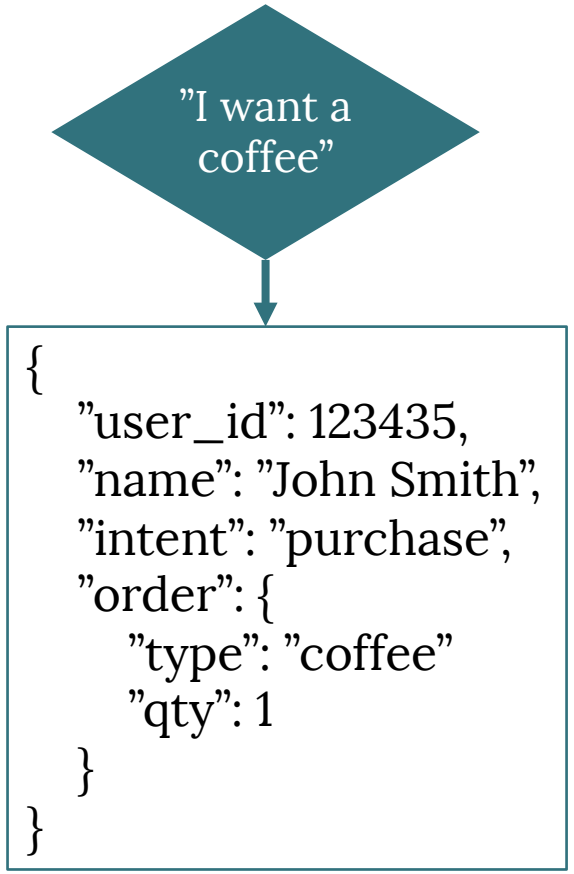
WIT.AI APPROACH

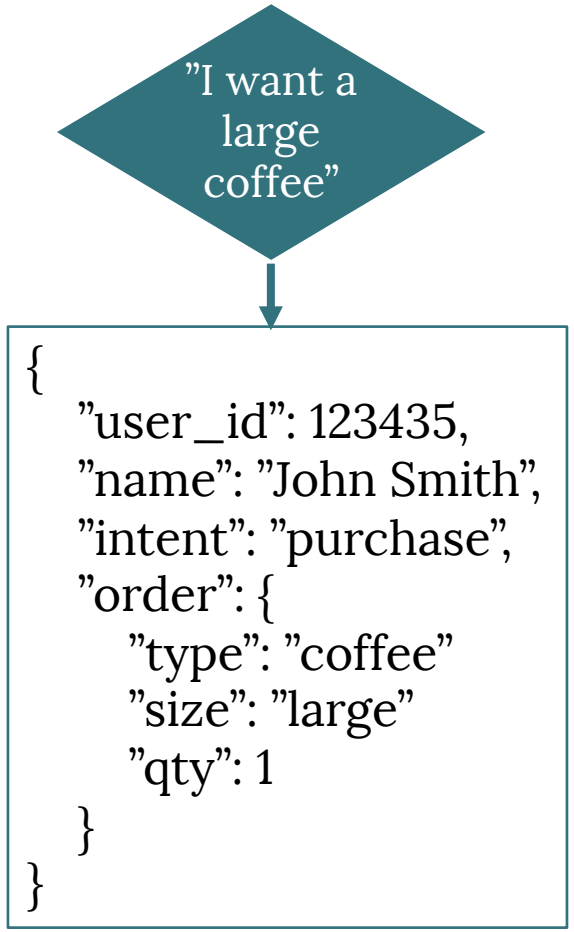
- Stories to define ”rules”/”heuristics” for the conversation
- Context to define current state

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

SC5

CONTEXTS





SC5

NATURAL LANGUAGE PROCESSING

”How is the weather tomorrow?”


```
{
    ”user_id”: 123435,
    ”name”: ”John Smith”,
    ”intent”: ”weather”,
    ”date”: ”2016-10-27T08:11:42.222Z”
}
```


”What’s the weather in London?”


```
{
    ”user_id”: 123435,
    ”name”: ”John Smith”,
    ”intent: ”weather”,
    ”location”: ”London"
}
```


SC5

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
}
```


SC5

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

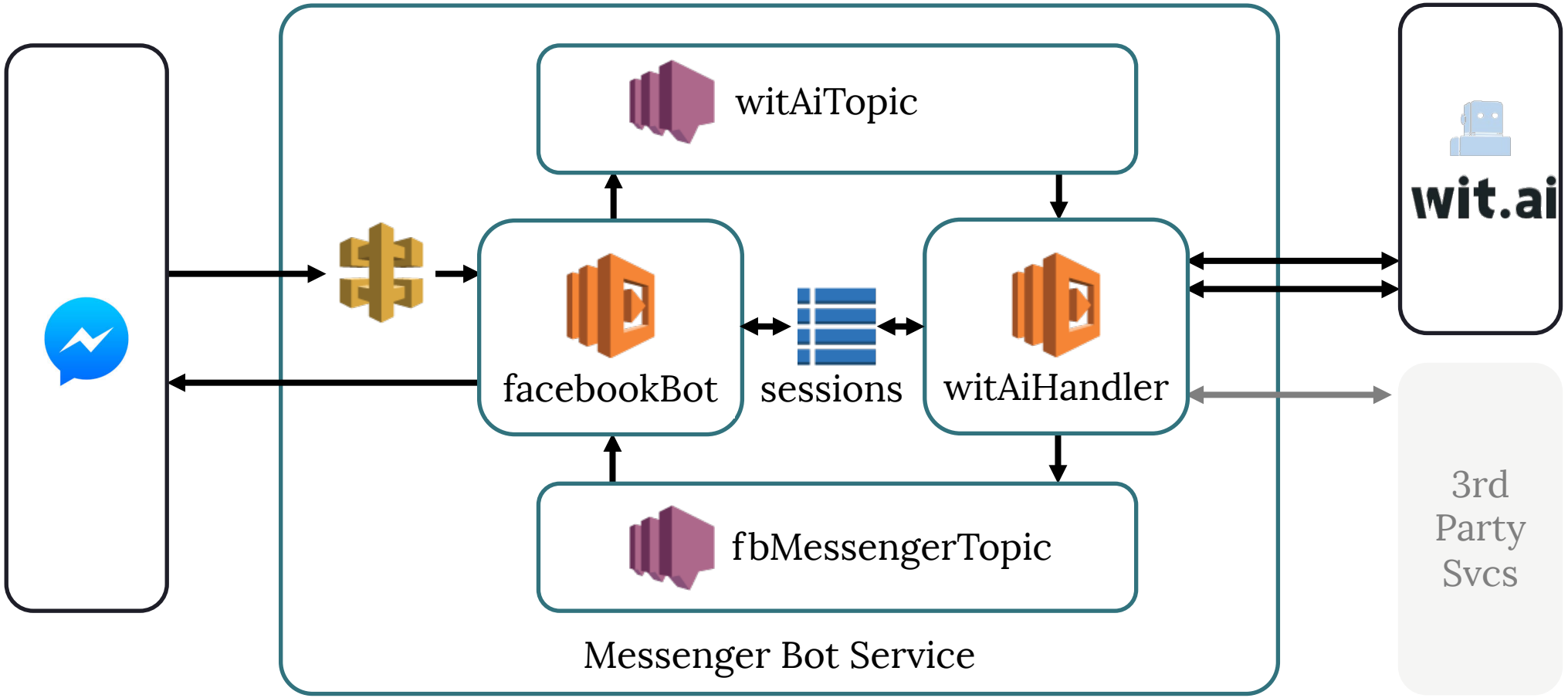
https://github.com/wit-ai/node-wit

OUR BOILERPLATE

serverless-messenger-boilerplate and its documentation available at

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

BOILERPLATE ARCHITECTURE



SC5

BOILERPLATE STRUCTURE

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

SC5

.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* and *.env*, 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

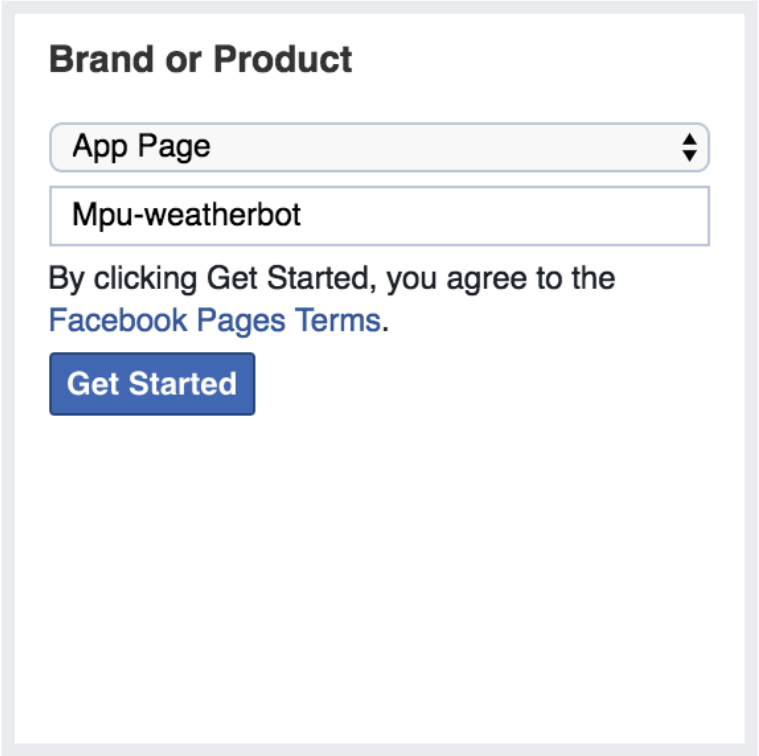
```
> sls deploy
```

SC5

SETUP FACEBOOK PAGE AND APPLICATION

CREATE A FACEBOOK PAGE

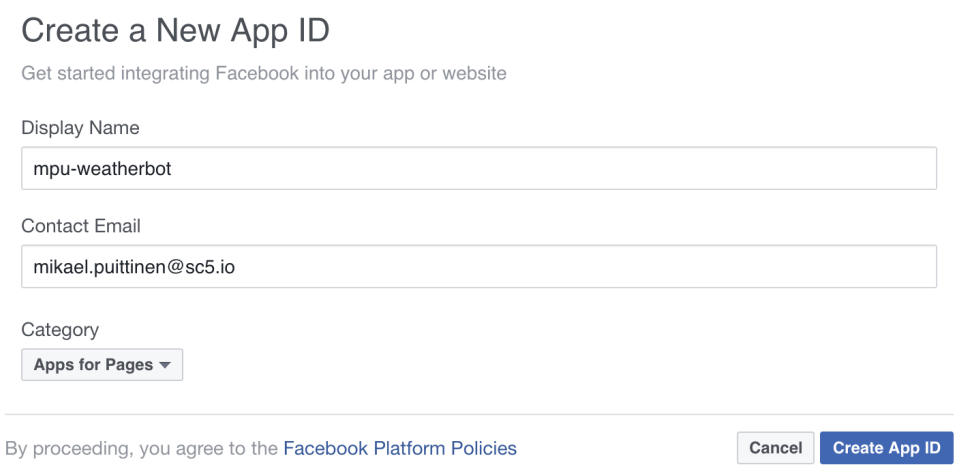
- Go to http://facebook.com/pages/create
- Create new page
- Skip all following steps



SC5

CREATE FACEBOOK APPLICATION

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

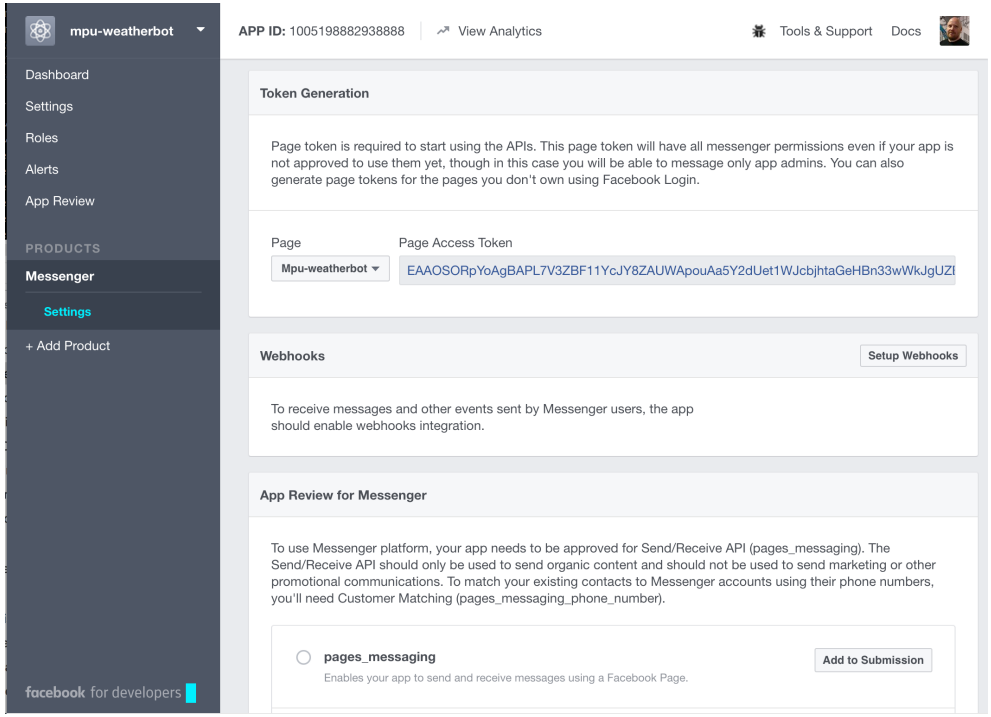


SC5

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

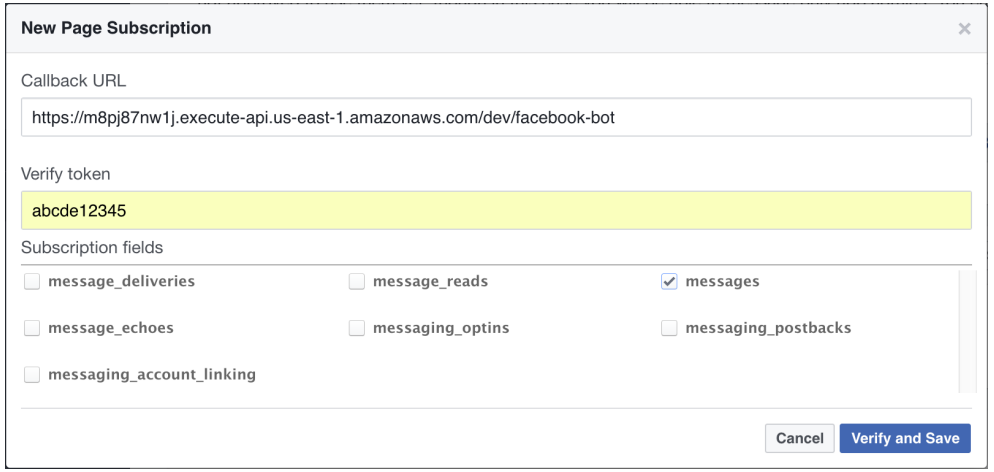
```
> sls deploy
```



SC5

SETUP MESSENGER WEBHOOK

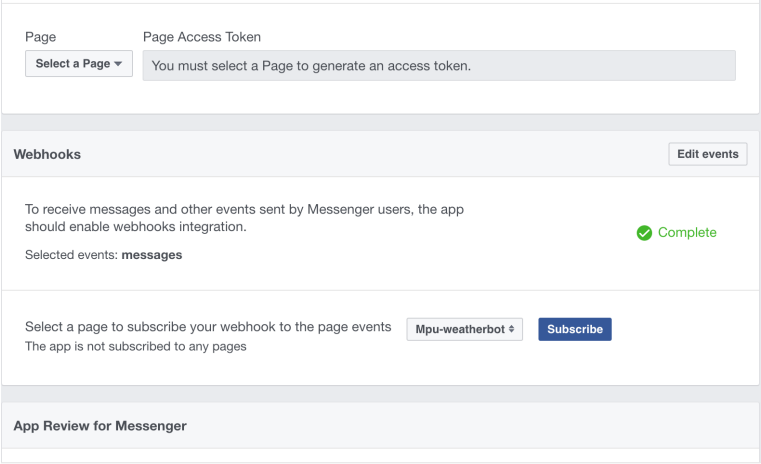
- Initiate ”Setup Webhooks”
- Enter the endpoint URL that you got during deployment
- Enter your verify token from *.env* (*FACEBOOK_VERIFY_TOKEN*)
- Verify and Save



SC5

SUBSCRIBE THE WEBHOOK TO THE FACEBOOK PAGE

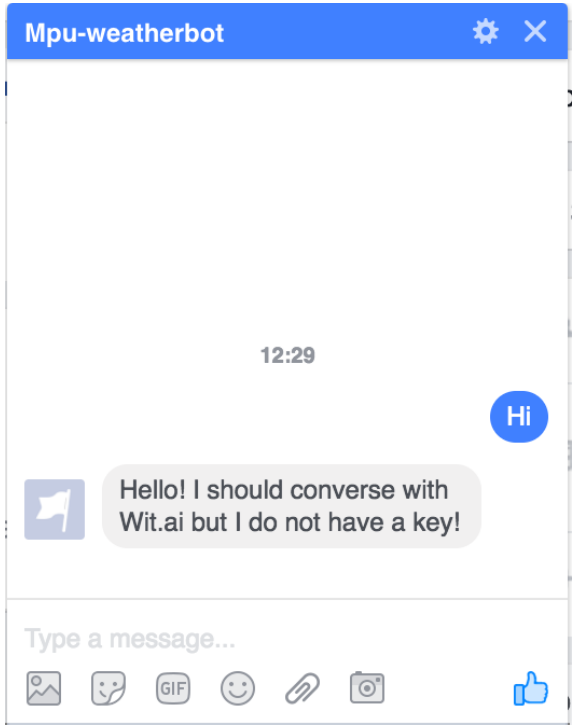
1. Under ”Webhooks”, select your page and subscribe



SC5

TRY IT

- Send message to your Facebook page



FACEBOOK APPLICATION ACCESS

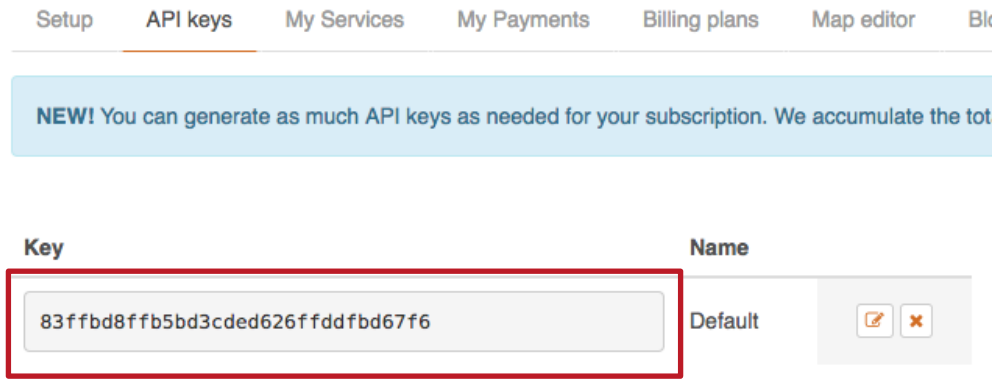
Access to unapproved applications is limited to developers only.

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*

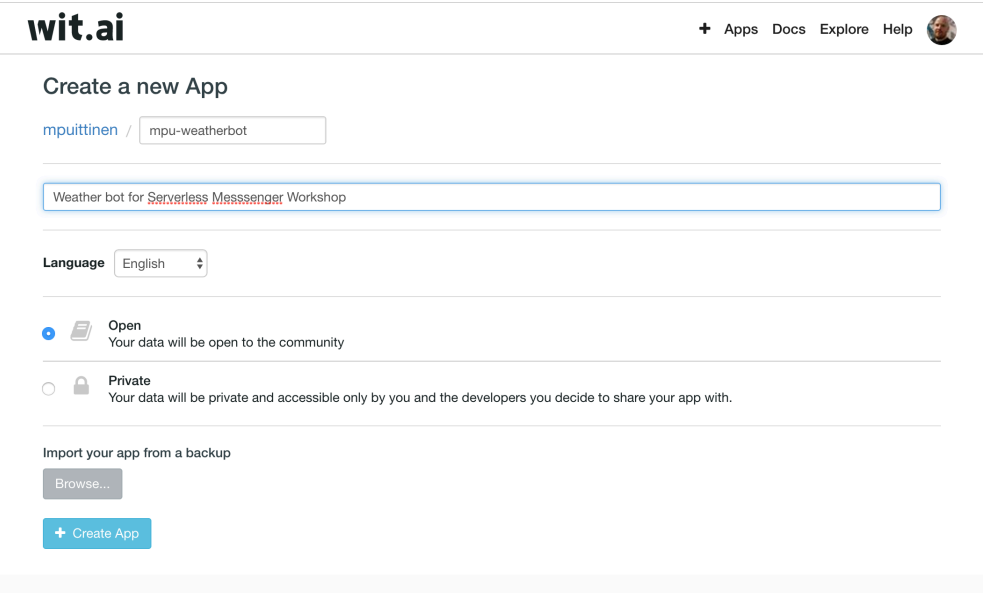


SC5

SETUP WIT.AI

REGISTER + CREATE WIT.AI APP

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

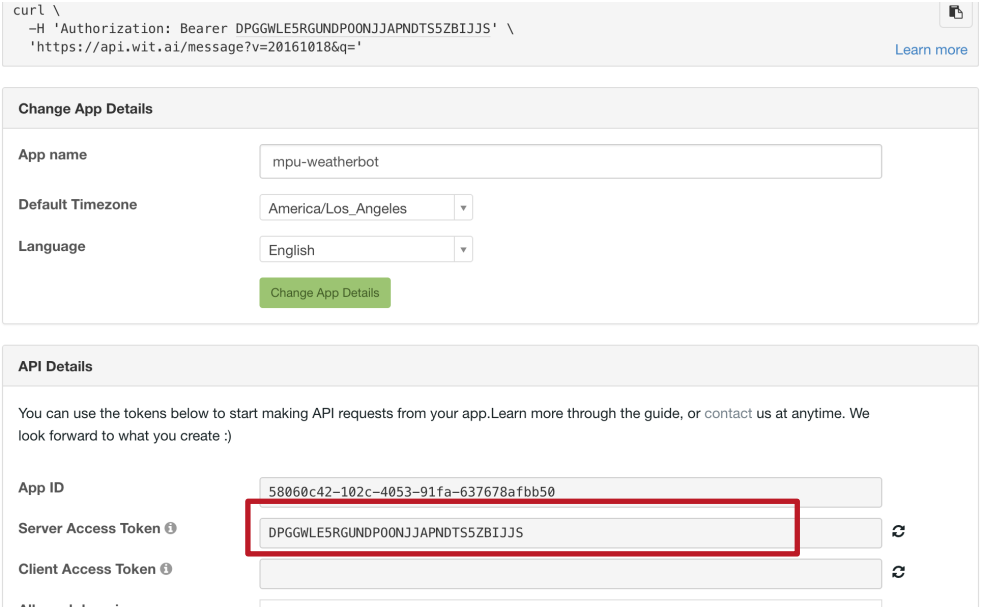


SC5

CONNECT THE ENDPOINT WITH WIT.AI

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

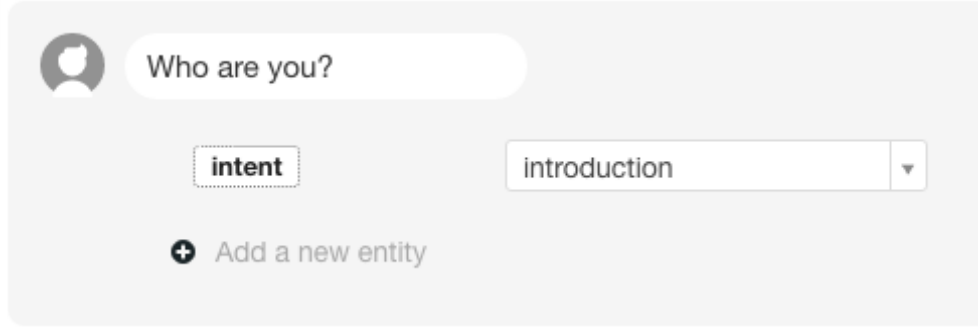
```
> sls deploy
```

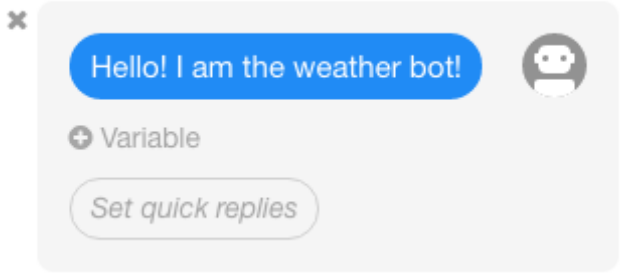


SC5

LET THE BOT INTRODUCE ITSELF

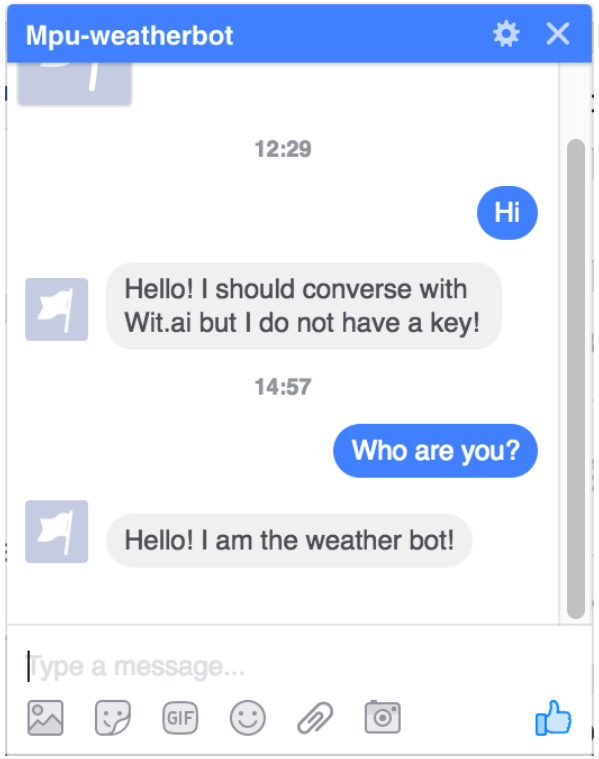
- Create a new story
- Add user input and intent
- Add a response with "Bot Sends"
- Remember to save the story





SC5

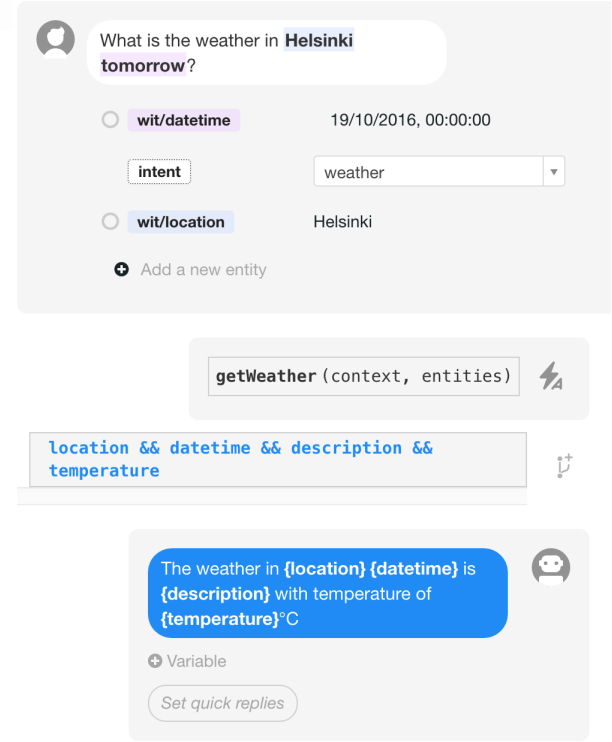
TRY IT!


Mpu-weatherbot
12:29
Hi
Hello! I should converse with Wit.ai but I do not have a key!
14:57
Who are you?
Hello! I am the weather bot!
Type a message...
GIF

SC5

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



SC5

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
```

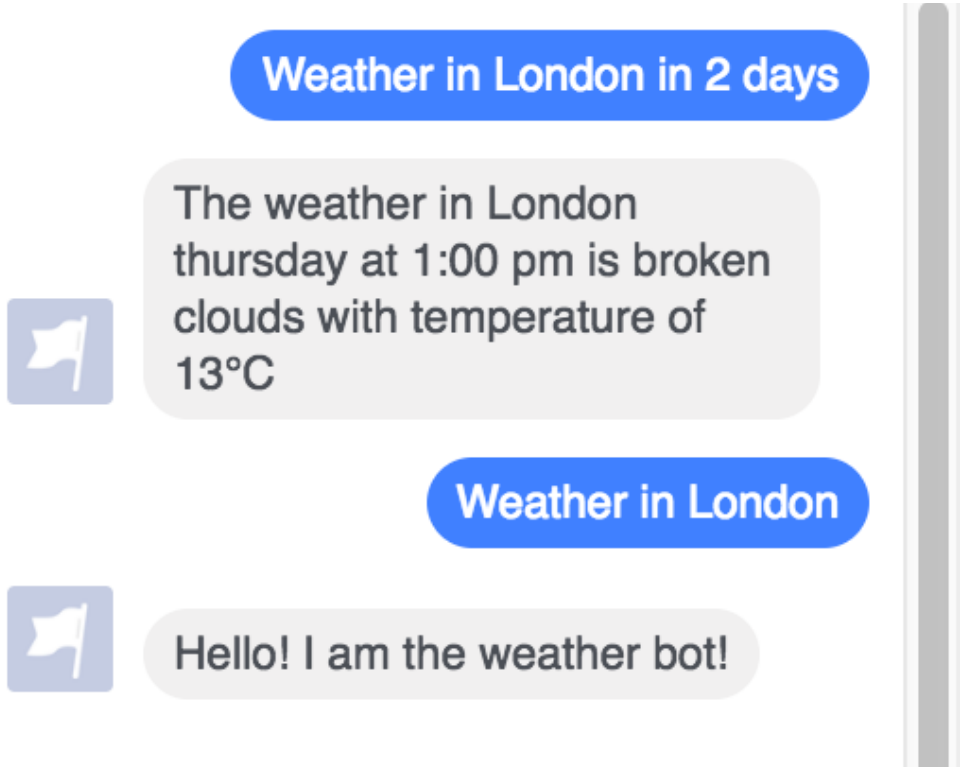
```javascript
const moment = require('moment');
const weather = require('./weather');

const actions = {
  getWeather: (data) => new Promise((resolve, reject) => {
    const context = data.context;
    const entities = data.entities;

    const missingLocation = entities.location === undefined;
    const location = entities.location ? entities.location[0].value : null;
    const datetime = entities.datetime ? entities.datetime[0].value : null;

    if (missingLocation) {
      const contextData = Object.assign({}, context, { missingLocation });
      resolve(contextData);
    } else if (datetime) {
      weather.forecastByLocationName(location, datetime)
        .then((weatherData) => {
          const contextData = Object.assign({}, context, weatherData);
          if (datetime) {
            Object.assign(contextData, { datetime: moment(datetime).calendar().toLowerCase() });
          }
          resolve(contextData);
        })
        .catch(reject);
    } else {
      weather.weatherByLocationName(location)
        .then((weatherData) => {
          const contextData = Object.assign({}, context, weatherData);
          if (datetime) {
            Object.assign(contextData, { datetime: moment(datetime).calendar().toLowerCase() });
          }
          resolve(contextData);
        })
        .catch(reject);
    }
  }),
```

SC5

TRY IT!

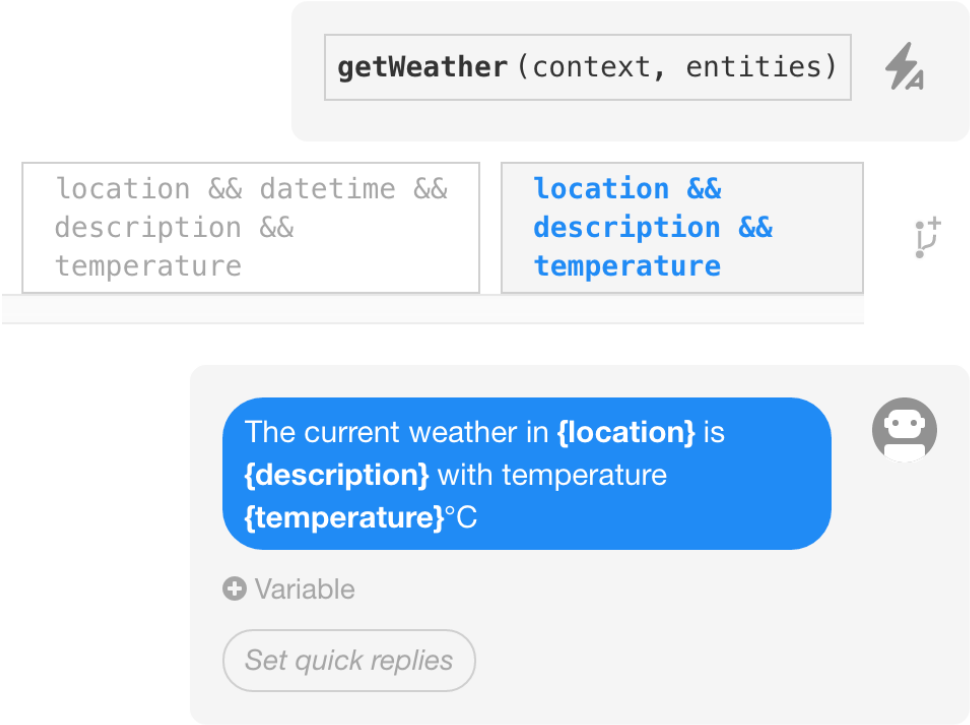
- An inquiry with date and location information works!
- But if we drop out the date, we do not get the right response.

Let's fix that!



FORK THE STORY WITHOUT A DATE

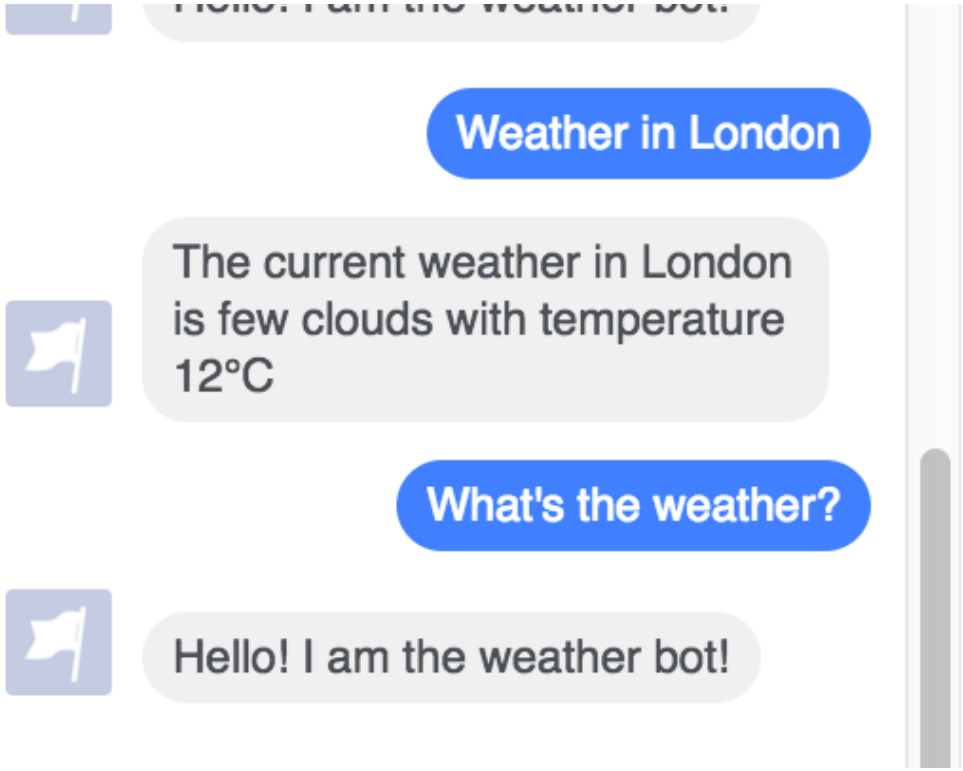
- Click the fork icon next to the context keys
- Create new context key *location && description && temperature* (without *datetime*)
- Create new response
- Save



SC5

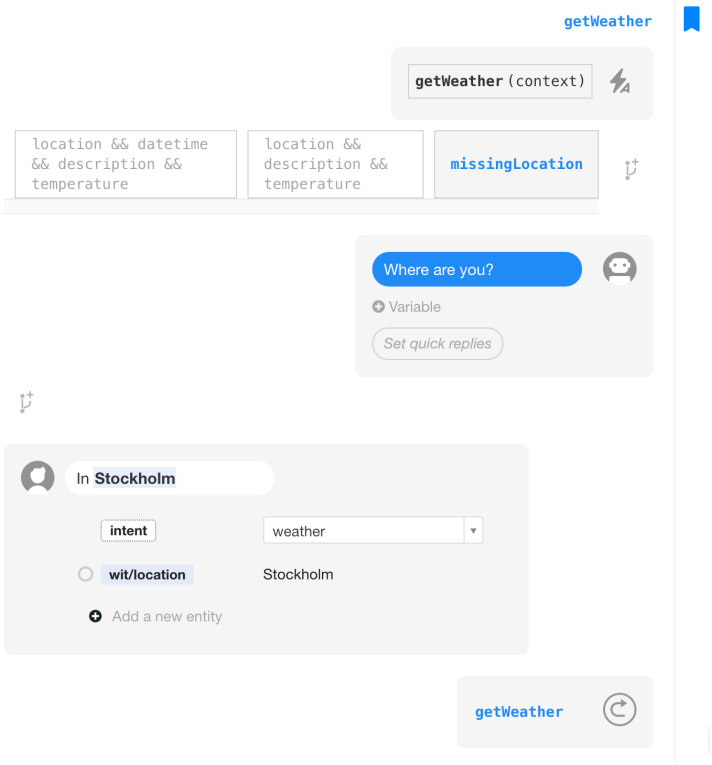
TRY IT!

- 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

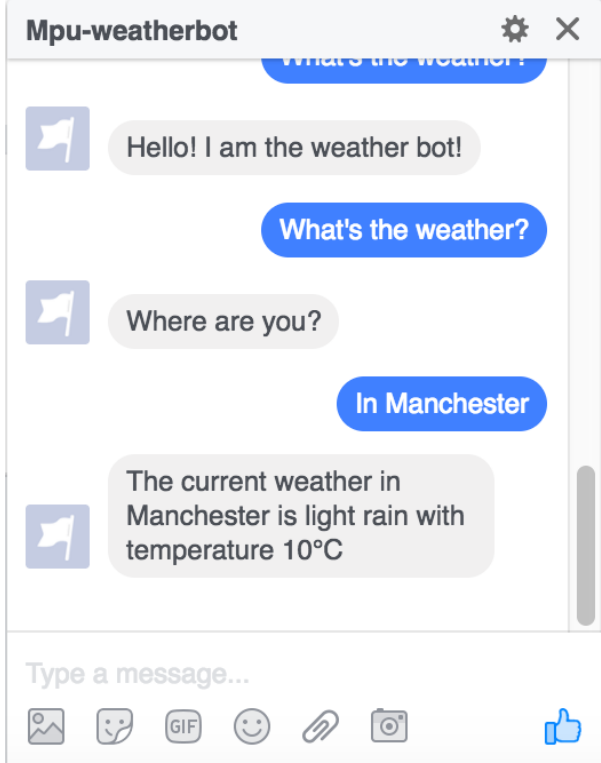
- Click the bookmark icon next to the *getWeather* action and create a bookmark *getWeather*
- Create new fork with context key *missingLocation*
- Add a bot response for asking the location
- Add a user response and set *intent* and *wit/location* entities
- Add Jump to *getWeather* bookmark created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



SC5

TRY IT!

IT WORKS!

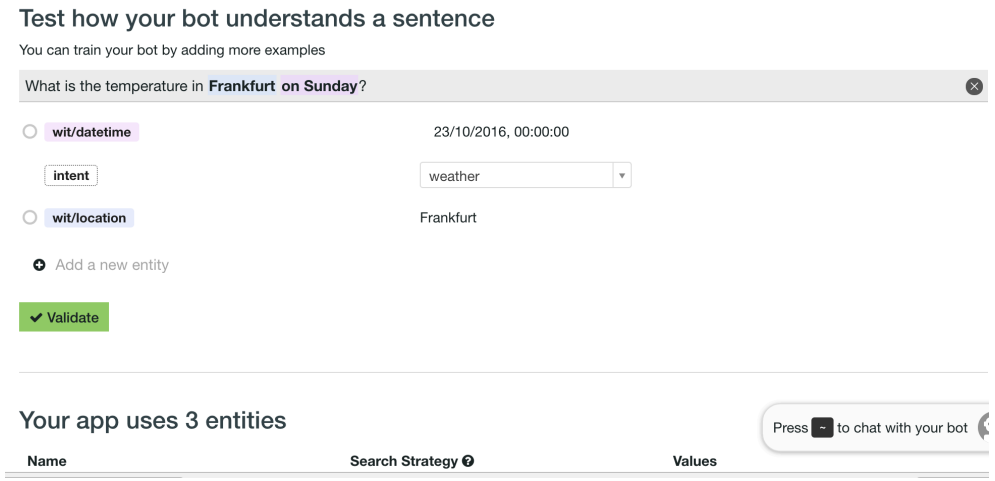


SC5

TRAINING THE BOT

WIT.AI UNDERSTANDING

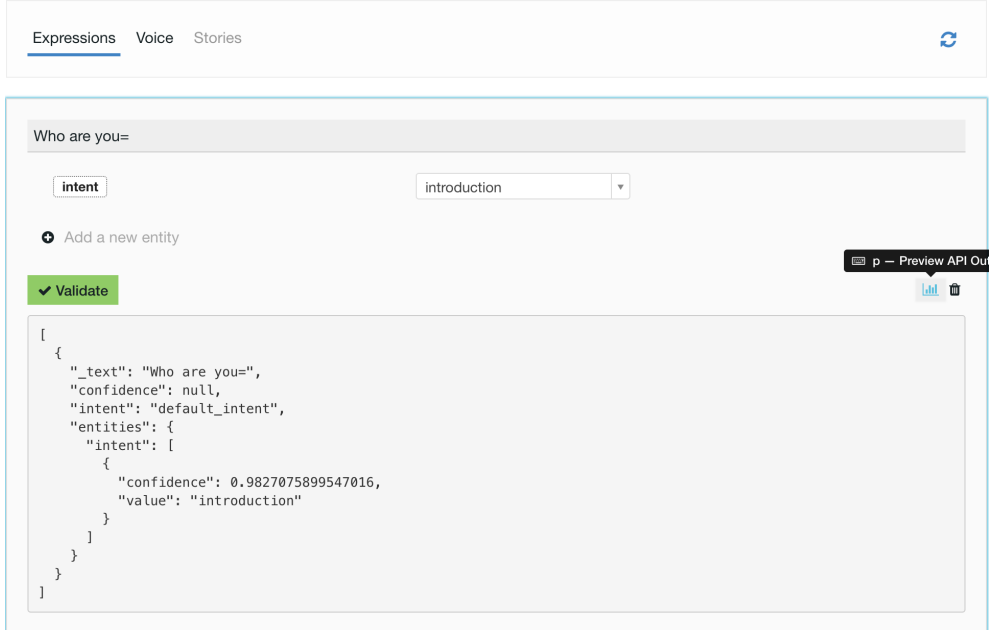
- Use Wit.ai understanding to train model
- Enter expressions, set entities and validate to improve accuracy



SC5

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



SC5

SC5

THANK YOU!

mikael.puittinen@sc5.io

@mpuittinen