WEB DESIGNER TEMPLATE
Updated 1 December 2015

Name: Tutorial_5page_dev01
Description: 5 pages | PAGE 1: Title, image and standfirst | PAGE 2: Intro, screenshot box, steps and blob | PAGE 3: steps, screenshot box and blob | PAGE 4: indetail and steps | PAGE 5: indetail/screenshot and steps,

About this tutorial:
This tutorial has a total of 21 steps which will flow through 4 pages, starting at Page 2 and goes to Page 5. There are no specific number of steps per page, but will be designed to best fit with code removed or put in in order to do so. The steps appear as one continuous flow of text with step code. Step images and captions appear in separate boxes. An indetail box provides an extended tip with an image to illustrate.

NOTE ABOUT CODE
PLEASE READ THIS CAREFULLY:

- A working example that can be viewed live should be provided with this tutorial.
- Code should be provided with this tutorial as well, inside of this template but it must also be collated as a whole into a separate document.

/** TEMPLATE ****/**

NOTE: Any RED text in this template is purely an instruction and should be stripped out once followed, along with any word count indicators – thank you. Make sure all images are labelled as instructed by the //pic:// tags and are submitted in .tiff formats wherever possible.

**PLEASE RESPECT THESE WORD COUNTS – if text is supplied that is considerably/ consistently under these targets it will be sent back. **

PAGE 01

//AUTHOR NAME//
Tim Stone

//AUTHOR BOX//
//text// 220 characters with spaces
Tim is Front End Developer at Lush with four years prior experience working with digital agencies. This month we're going to create a chat bot which integrates with Slack using SuperScript.js and some music APIs.

//pic: mugshot.tiff//

//TITLE// 50 characters with spaces
Join the bot revolution by creating a Slack bot

//STAND FIRST// 100 characters with spaces

Build a bot that integrates with Slack, gets the user's most recently listened to track, and find music to play.

//MAIN IMAGE//

//INTRO//
//text// 1250 characters with spaces
Chat bots have risen in popularity thanks to services like Slack which allow easy integration into them. Many organisations are now benefitting from the production increases they bring and are opening up the field of conversational UIs, interfaces which purely rely on dialogue.

These bots are quite complicated to start. You need to know some natural language processing, programming, working with APIs, all in all the niche of developers that fill all those requirements is pretty small. This is where (as with every case like this) frameworks come in. SuperScript (http://superscriptjs.com) is a framework for building chat bots harnessing many advanced features while keeping it simple enough to get started without a PhD. Not only that but it has example clients using Slack, telnet, Twilio, and WebSockets.

SuperScript has been around for a while but isn't widely known. Founder Rob Ellis (@rob_ellis) co-created PhoneGap, worked at Joyent on Node.js cloud projects and now works at Slack; he has also actively maintained SuperScript since 2014.

Our bot will be a glimpse at what's possible and focused on music. On the code included there is also functionality to get a recommended similar artist to the one you ask for using similar techniques.

PAGE 02

//STEP BY STEP//

//step01//
//title 15-30 characters with spaces //
Create a Slack bot
//text 200-300 characters with spaces not including code//
If you haven't got a Slack team already then head to https://slack.com/create to create one. Once you've setup your team you can create a bot at https://<slack_name>.slack.com/services/new/bot. This will give you a token which we'll use later on to authenticate the bot.

//step02//
//title 15-30 characters with spaces //
Install MongoDB
//text 200-300 characters with spaces not including code//
Before we install anything you need to have MongoDB on your machine. Instructions for most platforms can be found at https://docs.mongodb.com/manual/installation/. On OS X, if you have Homebrew, the process is very simple; there's a manual installation procedure but Homebrew is highly recommended!

//code maximum 10 lines – each line is 40 characters with spaces //

```
$ brew update
$ brew install mongodb
```

//step03//
//title 15-30 characters with spaces //
Run Mongo daemon
//text 200-300 characters with spaces not including code//
In a new CLI instance start up MongoDB. Later on it helps to be able to see what's happening in the database, http://mongodb-tools.com/ has a great list of GUIs for various platforms. MongoHub is a good choice if on OS X but a number of web clients are available too.

//code maximum 10 lines – each line is 40 characters with spaces //
```
$ mongod
```

//step04//
//title 15-30 characters with spaces //
Install SuperScript
//text 200-300 characters with spaces not including code//
SuperScript needs to be installed globally to create new bots from the CLI. We recommend using Node.js 4.4.x as 6.2.x had issues installing some of the dependencies on OS X. This process will take a while as there are plenty of packages to download and native extensions to compile!

//code maximum 10 lines – each line is 40 characters with spaces //
```
$ npm install superscript –g
```

//step05//
//title 15-30 characters with spaces //
Generate the bot
//text 200-300 characters with spaces not including code//
Once installed, SuperScript installs a program called bot-init. As the name suggests you can use this to quickly setup a bot. We're calling ours 'muzak' but feel free to name it however you wish. We pass a client parameter through which can be either Slack or telnet. Finally, install the dependencies (again!).

//code maximum 10 lines – each line is 40 characters with spaces //
```
$ bot–init muzak ––client slack
$ cd muzak
$ npm install
```

//step06//
//title 15-30 characters with spaces //
Insert auth token
//text 200-300 characters with spaces not including code//
Open the file called server.js in the muzak folder. Line 4 has a space for you to paste the API token that you received from Slack in the first step. Or you can click on Team Directory, select the bot and it'll have the token under Integration Settings.

//code maximum 10 lines – each line is 40 characters with spaces //

```
// Auth Token - You can generate your token from
// https://<slack_name>.slack.com/services/new/bot
var token = "YOUR_TOKEN";
```

//BLOB//
//title 15-20 characters with spaces//
Debug your gambits
//text 150-200 characters with spaces//
Run it normally with `$ node server.js` or `$ DEBUG=*,-Utils node server.js` to launch it in debug mode to see exactly how SuperScript matches gambits.

//screenshots//
//pic: SS_SS_slack_step01.png//
//caption//120-150 characters with spaces
The first step is to create the bot so we can get an API token. If you want to publish a bot the process is slightly different.

//pic: SS_SS_homebrew_step02.png//
//caption//120-150 characters with spaces
Homebrew is a software package manager for OS X. It's great for installing dependencies like MongoDB without having to build from source.

//pic: SS_SS_hello_step07.png//
//caption//120-150 characters with spaces
Our first gambit works! You may notice that there's a small delay built-in to the response, this is to simulate someone typing.

PAGE 03

//step07//
//title 15-30 characters with spaces //
Your first gambit
//text 200-300 characters with spaces not including code//
The crux of all bots are their inputs and outputs. In the generated folder called 'topics' is a file called main.ss. These SS files will contain the instructions for how your bot communicates with someone. The '+' indicates *input* and the '-' indicates the response. When the bot receives 'Hello' it responds 'Hi there!'.

//code maximum 10 lines – each line is 40 characters with spaces //
```
+ Hello
- Hi there!
```

//step08//
//title 15-30 characters with spaces //

Word expansion and parsing
//text 200-300 characters with spaces not including code//
There are many ways to greet each other and thankfully we don't have to specify each one, we can use *word expansion* to write one word and capture many. The tilde expands the word and emohello is a special keyword which captures all the forms of hello. Parse the gambits each time you change them by using the parse command.

//code maximum 10 lines – each line is 40 characters with spaces //
```
+ ~emohello
- Hi there!
// in CLI
$ parse -f
```

//step09//
//title 15-30 characters with spaces //
Multiple replies
//text 200-300 characters with spaces not including code//
We'll use those concepts to do the same for saying thank you. Note that we can specify *many* replies and it'll randomly choose one. In the same way that we vary between saying 'thank you', 'thanks', 'cheers' etc. having a bot pick from a number of replies enhances its naturalness.

//code maximum 10 lines – each line is 40 characters with spaces //
```
+ ~emothanks
- You're welcome
- No problem
- My pleasure
```

//step10//
//title 15-30 characters with spaces //
Multiple trigger words
//text 200-300 characters with spaces not including code//
We want to capture and save some text from a sentence when the user says their name/username. [a|b|c] is a list of alternates but one of them *must* be present to match it. *1 is a wildcard (any text) of 1 word length, this is what is captured in <cap>. ^save is a built-in function which we call to save the key, 'firstName', with the string from <cap>.

//code maximum 10 lines – each line is 40 characters with spaces //
```
+ My [user|name|username|login] is *1
- {keep} Hello <cap>! ^save(firstName, <cap>)
```

//step11//
//title 15-30 characters with spaces //
Trigger plugin function
//text 200-300 characters with spaces not including code//
Now that we have their username we can make requests about them. If they say something like 'What did I last listen to?' or 'What are my recent tracks?' we can get that information for them using a plugin.

//code maximum 10 lines – each line is 40 characters with spaces //
```
+ * (recent|latest|last|listening|listened) *
- {keep} ^getRecentTracks()
```

//screenshots//
//pic: SS_SS_apis_step12.tif//
//caption//120-150 characters with spaces
The API documentation for the Last.fm and Spotify wrappers follow the same as the official documentation so head there to find out what you can pass.

//pic: SS_SS_recent_step14.png//
//caption//120-150 characters with spaces
The bot calls the plugin to get the data from Last.fm, we could also use filters to capture the username in a different reply if we haven't got it.

//pic: SS_SS_play_step17.png//
//caption//120-150 characters with spaces
The 'play' gambit searches Spotify for matches, the code on the repo also includes a contextual 'Play it' which works with the recently played gambit.

PAGE 04

//BLOB//
//title 15-20 characters with spaces//
Writing gambits
//text 150-200 characters with spaces//
The project wiki has the most comprehensive set of documentation (http://bit.ly/29iSe0q) but there are plenty of areas that remain obscure, looking at unit tests and their Slack channel can help.

//step12//
//title 15-30 characters with spaces //
Configure Last.fm API
//text 200-300 characters with spaces not including code//
Visit http://www.last.fm/api and make an API account with them. This will give you an API key and secret, similar to Slack's process. For ease, we're going to use the lastfmapi module on npm, $ npm install lastfmapi --save.

//code maximum 10 lines – each line is 40 characters with spaces //
```
var LastfmAPI = require('lastfmapi');
var _ = require('lodash');

var lfm = new LastfmAPI({
    'api_key': 'YOUR_API_KEY',
    'secret': 'YOUR_API_SECRET'
});
```

//step13//
//title 15-30 characters with spaces //
Get recent tracks
//text 200-300 characters with spaces not including code//

SuperScript plugins are simply files in the plugin folder which export a function. Each function receives a callback function which you invoke when you're ready to pass back a response. In this case we get the user's firstName variable and use it to create an API request.

```
//code maximum 10 lines – each line is 40 characters with spaces //
exports.getRecentTracks = function (cb) {
    this.user.getVar('firstName', function (e, name) {
        lfm.user.getRecentTracks({
            user: name,
            limit: 1
        }, function (err, data) {
            // next step
        });
    });
};
```

//step14//
//title 15-30 characters with spaces //
Respond with callback
//text 200-300 characters with spaces not including code//
If there's an error we can send a response back and use Slack's emoji syntax to add more personality. If we've got a response with a track then we can format that data into a sentence to respond with.

```
//code maximum 10 lines – each line is 40 characters with spaces //
if (err) {
    cb(null, 'Sorry, I couldn\'t find you :slightly_frowning_face:');
} else {
    cb(null, 'Your most recent track is ' + data.track[0].name + ' by '
+ data.track[0].artist['#text'] + '.');
}
```

//step15//
//title 15-30 characters with spaces //
Play me a song
//text 200-300 characters with spaces not including code//
Now to create a new trigger, this time to find/play a song, anything from 'Play Beethoven' to 'Find Le Tigre' to 'Play North Borders – Bonobo' will be matched. In the response we invoke the searchTracks plugin and pass two arguments, the captured text, and a Boolean to denote it's the first time we're calling it.

```
//code maximum 10 lines – each line is 40 characters with spaces //
+ (Play|Find) *(1-20)
- ^searchTracks(<cap1>, true)
```

//step16//
//title 15-30 characters with spaces //
Harness the Spotify API
//text 200-300 characters with spaces not including code//
Install spotify-web-api-node ($ npm i spotify-web-api-node --save). Again, the callback has been appended to the arguments. We've captured everything past 'Play' or 'Find' up to 20 words, unfortunately we can't perform a generic search against

Spotify so for simplicity I'm assuming that they want a track.

//code maximum 10 lines – each line is 40 characters with spaces //

```
var SpotifyWebApi = require('spotify-web-api-node');
var _ = require('lodash');
var spotifyApi = new SpotifyWebApi();
exports.searchTracks = function (text, first, cb)
    spotifyApi.searchTracks(text)
    .then(function (data) {
        // next step
    });
};
```

//IN DETAIL//
//pic: xxx // Image to illustrate
//title 25-30 characters with spaces//
A short glossary of NLP terms
//text 700 characters with spaces//
Editor: there are some references to a SuperScript Editor, this is an editor from the creator but is not currently maintained.
Lemma: lemmatising essentially simplifies words within the context of other words.
Gambit: a rule telling the bot what and how to respond to, includes the input and reply.
POS: parts of speech, breaking each part of a sentence up into nouns, verbs, etc. This is usually combined with a tagger to tag each part of speech with its type.
Stemming: reduces inflected and derived words to their root form, e.g. writing to write.
Topic: contain one or more gambits, can be used to group types of gambit and can flow from one topic to another based on input.
Trigger: a rule which analyses the user input and tries to match specific conditions.

PAGE 05

//step17//
//title 15-30 characters with spaces //
Contextual responses
//text 200-300 characters with spaces not including code//
If a track is found and it's the first time this has been run then return the Spotify URI. Slack automatically expands this with its Spotify integration to show an inline player with the track information. If this isn't the first run through then it grabs a random track from the array and returns that instead.

//code maximum 10 lines – each line is 40 characters with spaces //

```
if (data.body.tracks.items.length) {
  if (first === 'true') {
    cb(null, 'I found this, any good? ' +
data.body.tracks.items[0].uri);
  } else {
    cb(null, 'How about this, any better? ' +
_.sample(data.body.tracks.items).uri);
  }
} else {
  cb(null, 'I couldn\'t find anything');
```

```
}
```

//step18//
//title 15-30 characters with spaces //
Building conversations
//text 200-300 characters with spaces not including code//
We can listen for certain inputs after specific gambits. For example, if we respond with 'How about this, any better?' and they reply yes then we can tailor a response. Note that we must specify the previous gambit with % followed by the gambit in question.

//code maximum 10 lines – each line is 40 characters with spaces //
```
+ ~yes
% Play *(1-20)
- Okay, good.
- Glad to hear it.
```

//step19//
//title 15-30 characters with spaces //
Using previous data
//text 200-300 characters with spaces not including code//
If they reply no (or something with the same sentiment) then our job isn't so easy. We can grab the previously captured text with <p1cap1> and tell searchTracks that this is a subsequent pass so grab a sample of the tracks, not the first one.

//code maximum 10 lines – each line is 40 characters with spaces //
```
+ ~no
% Play *(1-20)
- ^searchTracks(<p1cap1>, false)
```

//step20//
//title 15-30 characters with spaces //
Reusing saved data
//text 200-300 characters with spaces not including code//
When the user greets us we can use knowledge we've previously gained about the user with get(). You could make this smarter by using *filters* to vary the input based on if we have a first name saved or not.

//code maximum 10 lines – each line is 40 characters with spaces //
```
+ ~emohello
- ^get(firstName). Nice to see you again!
```

//step21//
//title 15-30 characters with spaces //
Saying farewell
//text 200-300 characters with spaces not including code//
To wrap up we want to be polite and say good bye if the user is gracious enough to do so. This is similar to previous examples but for completeness sake we reply 'Happy listening!'. We've seen SuperScript's power and flexibility but this is just the tip, be inspired and make bots!

//code maximum 10 lines – each line is 40 characters with spaces //
```
+ ~emogoodbye
- Happy listening!
```

//screenshots//
//pic: SS_SS_play-yes_step18.png//
//caption//120-150 characters with spaces
When the user says that they're happy with the song choice we acknowledge it in the same way you might show a tick if something has been synced with a server.

//pic: SS_SS_play-no_step19.png//
//caption//120-150 characters with spaces
If the user isn't happy with the song we chose then we can make the request again using the previous captured data and select a different track.

//pic: SS_SS_goodbye_step21.png//
//caption//120-150 characters with spaces
If the user signs off with a 'farewell' or 'goodbye' then we reply with Happy listening! Who knows, it could become a catchphrase.

[ENDS]