BotMan is the most popular chatbot development framework for PHP. You can use BotMan in your existing codebase or with any PHP frameworks.

There are different chatbot development frameworks available for Node Js, Python. BotMan is the only PHP framework that truly helps the developers to build chatbot using PHP.

If you are a Laravel developer, BotMan Studio is the one thing you are looking for. BotMan Studio is a bundled version of BotMan and Laravel.

How to create a chatbot using BotMan?

Let's take a look at BotMan Studio and build a simple Salon booking chatbot for the website.

Install and set up the BotMan Studio

First, make sure you have composer installed on your system. Now download the BotMan installer using Composer.

composer global require "botman/installer"

This will install the BotMan Studio on your system.

Make sure to place the $HOME/.composer/vendor/bin directory (or the equivalent directory for your OS) in your $PATH so the botman executable can be located by your system.

Type "botman" on your command line to see if installation works correctly. If you can see the output similar to the below image then BotMan Studio installation is a success.

Create a new chatbot project

Now let's use the BotMan Studio to create new chatbot. You can use 'botman new' command which will create a fresh BotMan Studio installation.

botman new coursebooking

The above command will create a new directory called "coursebooking" with the BotMan Studio installation with all of BotMan's dependencies already installed.

Test the installation works

Now start the development server using the

cd coursebooking

php artisan serve

That will start the development server at http://127.0.0.1:8000

If the installation is okay, you can see the welcome page from Botman.

Click the link "Tinker", there you can try a very simple chatbot. Say "Hi" to bot and the bot will reply with "Hello!".

BotMan chatbot development

Great job. Now you have everything ready to build your first chatbot.

Install the web driver

BotMan comes with many drivers for the developers. Drivers help us to connect the bot with messaging channels such as Facebook Messenger, Slack, Telegram, Twilio, Web etc.

As we are going to build a web-based chatbot, let's install the web driver.

php artisan botman:install-driver web

We can use WebDriver for adding the chatbot to our website or use the APIs. Web driver comes with APIs for sending and receiving messages. This helps us to build our own chat interfaces by leveraging the WebDriver APIs.

BotMan ships with a web widget that you can use out of the box. We can include the Javascript snippet to our welcome view (welcome.blade.php)

<script src='https://cdn.jsdelivr.net/npm/botman-web-widget@0/build/js/widget.js'></script>

Now, refresh your page and you can see the chat widget at the bottom right corner of the page. Say "Hi" to bot and the bot will reply with "Hello!". That's it.

BotMan chat widget

There are 2 important core concepts you should learn before starting building chatbot using the BotMan Studio - Hearing the Messages and Sending the Messages.

Like any other web application, chatbot receives some request, process it and send the responses back to the channels (Web, Messenger, Slack etc).

Hearing the Messages

$botman = resolve('botman');

$botman->hears('Hi', function ($bot) {

$bot->reply('Hello!');

});

You can even use a regular expression like below. It listens for either "Hi" or "Hello" anywhere in the incoming message.

$botman = resolve('botman');

$botman->hears('

Hi|Hello', function ($bot) {

$bot->reply('Hello!');

});

Instead of Closure, you can pass a class and method that will get called if the keyword matches. In the example below, startConversation method will get called if the user says "Hi".

$botman = resolve('botman');

$botman->hears('Hi|Hello', BotManController::class.'@startConversation');

Sending the Messages

You have already seen how to send a message using BotMan. BotMan can send messages in different ways - text, buttons, attachments, generic templates etc.

Let's see how to send a text message with two buttons.

$botman = resolve('botman');

$question = Question::create('Are you sure?')

->callbackId('agree')

->addButtons([

Button::create('Yes')->value('yes'),

Button::create('No')->value('no'),

]);

$botman->ask($question, function(Answer $answer) {

});

Chatbot conversation flows

We are going to build a simple salon booking chatbot which helps the website visitors to book the salon services through the chatbot from their website.

Every chatbot must have one or more conversation flows. Here is the expected conversation flow of chatbot we are going to build.

Welcome the visitor

Prompt the visitor to choose one of the services

Prompt the visitor to choose a date and time slot

Ask for the name, email, and mobile

Show the booking information

Build the chatbot dialogs

Now open routes/botman.php and change it so that when someone says 'Hi' or 'Hello', startConversation method in BotManController will get called.

<?php

use App\Http\Controllers\BotManController;

$botman = resolve('botman');

$botman->hears('Hi|Hello', BotManController::class.'@startConversation');

Create a new conversation class OnboardingConversation

php artisan botman:make:conversation OnboardingConversation

The above command will create the OnboardingConversation class at app/Conversations/

<?php

namespace App\Http\Conversations;

use BotMan\BotMan\Messages\Conversations\Conversation;

class OnboardingConversation extends Conversation

{

public function run()

{

//

}

}

Let's take a look at the BotManController class located at app/Http/Controllers/ and update the code so that it can use our new OnboardingConversation class.

<?php

namespace App\Http\Controllers;

use BotMan\BotMan\BotMan;

use Illuminate\Http\Request;

use App\Conversations\OnboardingConversation;

class BotManController extends Controller

{

public function handle()

{

$botman = app('botman');

$botman->listen();

}

public function tinker()

{

return view('tinker');

}

public function startConversation(BotMan $bot)

{

$bot->startConversation(new GreetingConversation());

}

}

Now open the conversation class GreetingConversation. It extends an abstract class Conversation. The run method is the starting point of the conversation and get's executed immediately.

<?php

namespace App\Http\Conversations;

use BotMan\BotMan\Messages\Conversations\Conversation;

class GreetingConversation extends Conversation

{

public function run()

{

//

}

}

Ask for the name, email, and mobile

Let's ask the visitor to enter his/her name. This is a very simple thing to do.

<?php

namespace App\Http\Conversations;

use BotMan\BotMan\Messages\Incoming\Answer;

use BotMan\BotMan\Messages\Conversations\Conversation;

class GreetingConversation extends Conversation

{

public function askService()

{

$greetings = "";

$time = date("H");

$timezone = date("e");

if ($time < "12") {

$greetings = "Good morning";

} else

if ($time >= "12" && $time < "17") {

$greetings = "Good afternoon";

} else

if ($time >= "17" && $time < "19") {

$greetings = "Good evening";

} else

if ($time >= "19") {

$greetings = "Good night";

}

$this->say('Hello '.$greetings);

$question = Question::create('What kind of Course you are looking for?')

->callbackId('select\_service')

->addButtons([

Button::create('Java')->value('Java'),

Button::create('Python')->value('Python'),

Button::create('php')->value('php'),

]);

$this->ask($question, function(Answer $answer) {

if ($answer->isInteractiveMessageReply()) {

$this->bot->userStorage()->save([

'service' => $answer->getValue(),

]);

}

$this->getAns('service',$answer->getValue());

});

//$this->bot->startConversation(new DateTimeConversation());

}

public function getAns($type,$value)

{

if($type == 'service')

{

$this->say('Great, you are selecting '.$value);

$this->askTeacher($value);

}

}

public function askTeacher($value)

{

if($value == 'Java')

{

$question = Question::create('select teacher')

->callbackId('select\_tech')

->addButtons([

Button::create('durga')->value('durga'),

Button::create('nagoor')->value('nagoor'),

Button::create('sriman')->value('sriman'),

]);

$this->ask($question, function(Answer $answer) {

if ($answer->isInteractiveMessageReply()) {

$this->bot->userStorage()->save([

'teacher' => $answer->getValue(),

]);

}

$this->say('Great, you are selecting '.$answer->getValue());

$this->askDate();

});

}

else if($value == 'Python')

{

$question = Question::create('select teacher')

->callbackId('select\_tech')

->addButtons([

Button::create('satish')->value('satish'),

Button::create('ganesh')->value('ganesh'),

]);

$this->ask($question, function(Answer $answer) {

if ($answer->isInteractiveMessageReply()) {

$this->bot->userStorage()->save([

'service' => $answer->getValue(),

]);

}

$this->say('Great, you are selecting '.$answer->getValue());

$this->askDate();

});

}

else if($value == 'php')

{

$question = Question::create('select teacher')

->callbackId('select\_tech')

->addButtons([

Button::create('swamy naidu')->value('swamy naidu'),

Button::create('samba')->value('samba'),

]);

$this->ask($question, function(Answer $answer) {

if ($answer->isInteractiveMessageReply()) {

$this->bot->userStorage()->save([

'service' => $answer->getValue(),

]);

}

$this->say('Great, you are selecting '.$answer->getValue());

$this->askDate();

});

}

}

public function askDate()

{

$availableDates = [

Carbon::today()->addDays(1),

Carbon::today()->addDays(2),

Carbon::today()->addDays(3) ];

$question = Question::create('Select joining date')

->callbackId('select\_date')

->addButtons([

Button::create($availableDates[0]->format('M d'))->value($availableDates[0]->format('Y-m-d')),

Button::create($availableDates[1]->format('M d'))->value($availableDates[1]->format('Y-m-d')),

Button::create($availableDates[2]->format('M d'))->value($availableDates[2]->format('Y-m-d')),

]);

$this->ask($question, function(Answer $answer) {

if ($answer->isInteractiveMessageReply()) {

$this->bot->userStorage()->save([

'date' => $answer->getValue(),

]);

$this->say('Great, you are selecting '.$answer->getValue());

$this->askTime();

}

});

}

public function askTime()

{

$question = Question::create('Select a prefare time')

->callbackId('select\_time')

->addButtons([

Button::create('9 AM')->value('9 AM'),

Button::create('1 PM')->value('1 PM'),

Button::create('3 PM')->value('3 PM'),

]);

$this->ask($question, function(Answer $answer) {

if ($answer->isInteractiveMessageReply()) {

$this->bot->userStorage()->save([

'timeSlot' => $answer->getValue(),

]);

$this->say('Great, you are selecting '.$answer->getValue());

$this->bot->startConversation(new OnboardingConversation());

}

});

}

/\*\*

\* Start the conversation.

\*

\* @return mixed

\*/

public function run()

{

$this->askService();

}

}

How does it work?

The run method will call the askName method when the conversation starts. Inside the askName it uses the 'ask' method which will output a question in the chatbot. The user response can be fetched using $answer->getText().

Now the bot wants to ask the 2nd and 3rd questions. So let's create methods - askTime(),askDate(),askTeacher() etc.

There are a few interesting things to notice.

First, I have used Laravel validation to validate the email. If the validation fails, the chatbot prompts the users to enter a valid email address. The method 'repeat' will ask the last asked question again.

$this->repeat('That doesn\'t look like a valid email. Please enter a valid email.');

userStorage method as the name suggests saving the current chatbot user information. Hence we will use userStorage method to store the booking information.

The bot will ask the question one after another. Here is the complete code.

Let's try our chatbot to see if that works. Go to http://127.0.0.1:8000, refresh the page and start the conversation by sending 'Hello'.

Botman Booking Chatbot

Prompt the visitor to choose one of the services

Now let's move to the next questions. This time the chatbot will ask the users to choose one of the services. Assume the salon offers 3 services - Hair, Spa, Beauty.

Create a new conversation class SelectServiceConversation

php artisan botman:make:conversation SelectServiceConversation

When we build more and more conversations, we want to connect conversations with one another. You can use the method $this->bot->startConversation(new AnotherConversation()) inside any conversation class to connect it with another conversation.

Open the OnboardingConversation conversation class and make the following changes.

public function askMobile()

{

$this->ask('Great. What is your mobile?', function(Answer $answer) {

$this->bot->userStorage()->save([

'mobile' => $answer->getText(),

]);

$this->askAddress();

//$this->say('Great!');

//$this->bot->startConversation(new SelectServiceConversation());

});

}

public function askAddress()

{

$this->ask('Great. What is your Address?', function(Answer $answer) {

$this->bot->userStorage()->save([

'address' => $answer->getText(),

]);

// $this->askAddress();

$this->say('Great!');

$this->bot->startConversation(new BookingConversation());

});

}

BotMan ships cool method to asking multiple choice questions - Question and Button classes.

isInteractiveMessageReply method detects if the user interacted with the message and clicked on a button or simply entered text.

Let's try our chatbot to see if that works. Go to http://localhost:8000, refresh the page and start the conversation by sending 'Hello'.

Prompt the visitor to choose a date and time slot

Great!. As what we have done before let's create one more conversation class to ask the users to choose a date and timeslot for the booking.

Don't forget to connect the DateTimeConversation conversation with SelectServiceConversation inside the askService method.

$this->bot->startConversation(new DateTimeConversation());

Finally, let's show the booking details back to the user using a new conversation class BookingConversation.

php artisan botman:make:conversation BookingConversation

Here is the BookingConversation class

<?php

namespace App\Http\Conversations;

use BotMan\BotMan\Messages\Incoming\Answer;

use BotMan\BotMan\Messages\Outgoing\Question;

use BotMan\BotMan\Messages\Outgoing\Actions\Button;

use BotMan\BotMan\Messages\Conversations\Conversation;

class BookingConversation extends Conversation

{

public function confirmBooking()

{

$user = $this->bot->userStorage()->find();

//dd($user);

$message = '-------------------------------------- <br>';

$message .= 'Course : ' . $user->get('service') . '<br>';

$message .= 'Teacher : ' . $user->get('teacher') . '<br>';

$message .= 'Name : ' . $user->get('name') . '<br>';

$message .= 'Email : ' . $user->get('email') . '<br>';

$message .= 'Mobile : ' . $user->get('mobile') . '<br>';

$message .= 'Address : ' . $user->get('address') . '<br>';

$message .= 'Date : ' . $user->get('date') . '<br>';

$message .= 'Time : ' . $user->get('timeSlot') . '<br>';

$message .= '---------------------------------------';

$this->say('Great. Your booking has been confirmed. Here is your booking details. <br><br>' . $message);

}

}