**演習 2: ボットによるヘルプ デスク チケットの送信 (Node.js)**

**概要**

この演習では、ボットに会話機能を追加して、ヘルプ デスク チケットの作成をユーザーに案内する方法を学習します。

[こちらのフォルダー](https://github.com/GeekTrainer/help-desk-bot-lab/blob/develop/Node/exercise2-TicketSubmissionDialog)内には、この演習のステップの完了結果として得られるコードを含むソリューションが入っています。このソリューションは、演習を進めるにあたってさらにヒントが必要な場合に、ガイダンスとして使用できます。ソリューションを使用するには、まず必ず npm install を実行してください。

**前提条件**

この演習を完了するには、以下のソフトウェアが必要です。

* [最新の Node.js と NPM](https://nodejs.org/en/download)
* [Visual Studio Code](https://code.visualstudio.com/download) (推奨) や Visual Studio 2017 Community 以上などのコード エディター
* [Bot Framework Emulator](https://emulator.botframework.com/) (en-US ロケールで構成されていることを確認してください)

**ラボ ノート**

サーバーとボットのコードが混在することで、app.js が多少長くなることに気づくかもしれません。これは、わかりやすくラボを進めるためのものです。より複雑なボットでは、通常、複数のファイルにボットを分割することになります。

**タスク 1: ボットへの会話の追加**

このタスクでは、ボットを変更して、いくつかアクションを実行する前に、ユーザーに一連の質問をします。

1. 前の演習から得られた app.js ファイルを開きます。または、[exercise1-EchoBot](https://github.com/GeekTrainer/help-desk-bot-lab/blob/develop/Node/exercise1-EchoBot) フォルダーからこのファイルを開くこともできます。
2. 以下のように、UniversalBot コネクターを作成するコードを更新します。ボットのメッセージ ハンドラーが、単一の関数ではなく、ウォーターフォールと呼ばれる関数の配列となっていることに気づくでしょう。ユーザーがこのボットにメッセージを送信すると、ウォーターフォールの最初の関数が呼び出されます。これはユーザーへのあいさつで、text() を使用して、まず、問題について説明することを求めます。SDK では、ユーザーからの入力の収集を簡単にするためのビルトイン プロンプトのセットを提供しています。

ユーザーの応答が、ウォーターフォールの 2 番目の関数に渡されます。この関数では、説明の内容を確認します。このカスケード形式の質問/応答シーケンスから、ウォーターフォールという名前が付いています。

また、応答はダイアログ データに保持されます。ダイアログ データは、1 つのダイアログ インスタンスの情報の保持に使用されます。これは、ダイアログのウォーターフォールのステップ間で一時情報を保存するために重要です。

var bot = new builder.UniversalBot(connector, [

(session, args, next) => {

session.send('Hi! I\'m the help desk bot and I can help you create a ticket.');

builder.Prompts.text(session, 'First, please briefly describe your problem to me.');

},

(session, result, next) => {

session.dialogData.description = result.response;

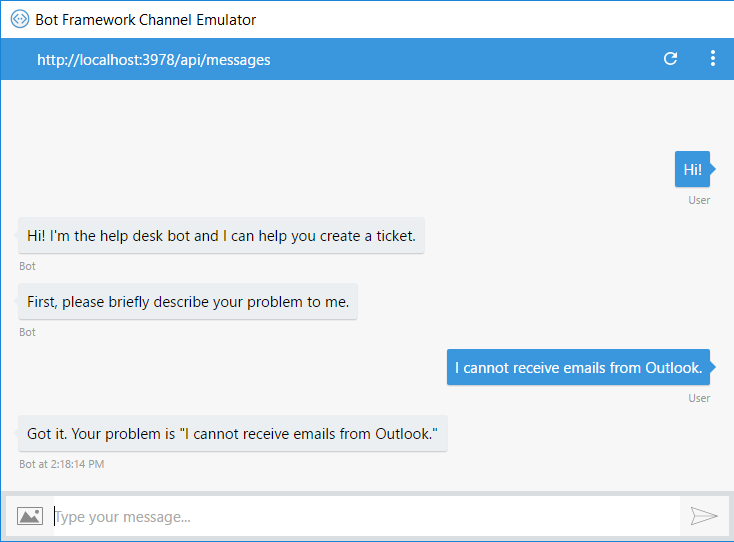
session.send(`Got it. Your problem is "${session.dialogData.description}"`);

session.endDialog();

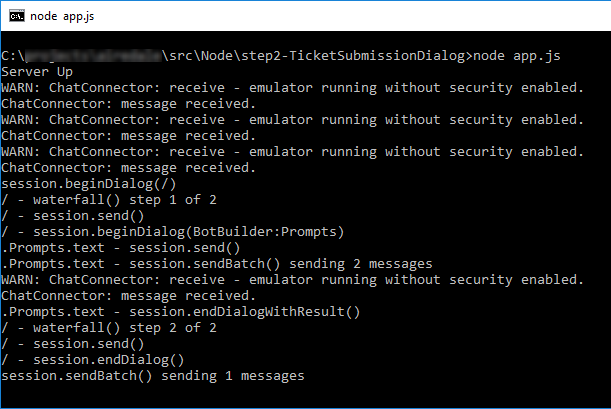
}

]);

1. コンソール (nodemon app.js) からアプリを実行し、エミュレーターを開きます。いつもどおりにボットの URL を入力し (http://localhost:3978/api/messages)、ボットをテストします。

[](https://github.com/GeekTrainer/help-desk-bot-lab/blob/develop/Node/images/exercise2-dialog.png)

1. コンソール ウィンドウでも、メッセージ ハンドラーが 1 つずつ実行される様子を確認できます。

[](https://github.com/GeekTrainer/help-desk-bot-lab/blob/develop/Node/images/exercise2-console.png)

**タスク 2: チケット詳細のプロンプト**

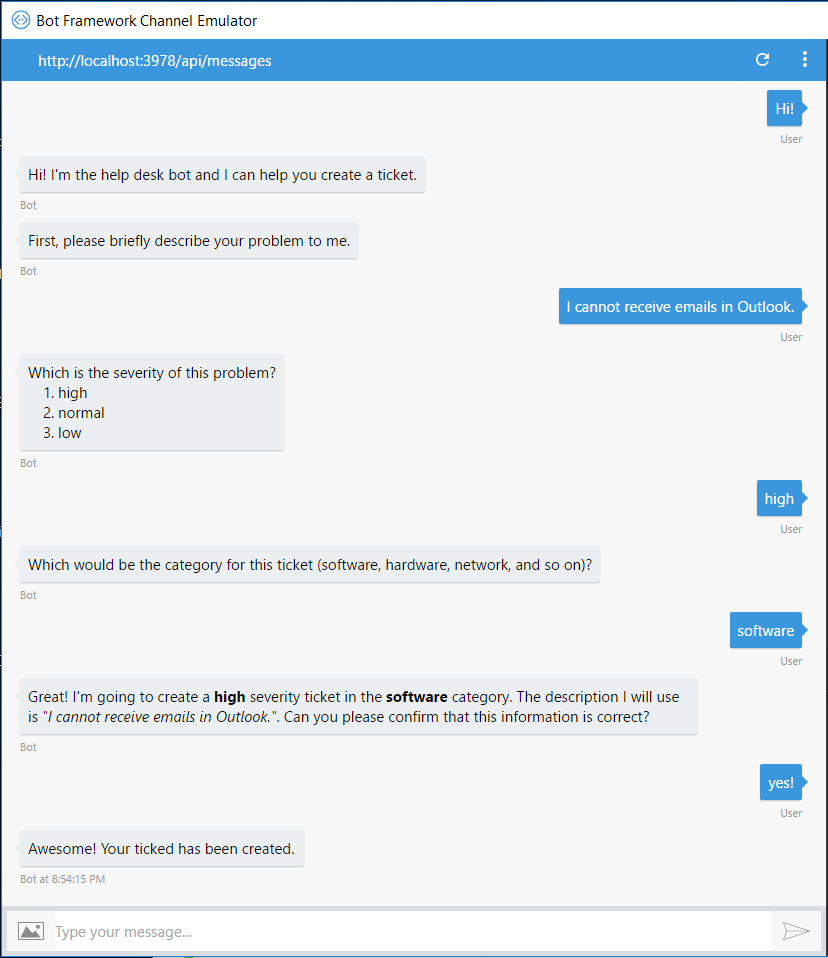
このタスクでは、さらに多くのメッセージ ハンドラーをボット ウォーターフォールに追加して、チケットのすべての詳細について尋ねます。

1. 以下のように、UniversalBot コネクターを作成するコードを更新します。これにより、2 種類のプロンプトが導入されます。
   * Prompts.choice(): チケットの重要度について尋ねます。
   * Prompts.confirm(): チケットの情報が正しいことを確認します。
2. var bot = new builder.UniversalBot(connector, [
3. (session, args, next) => {
4. session.send('Hi! I\'m the help desk bot and I can help you create a ticket.');
5. builder.Prompts.text(session, 'First, please briefly describe your problem to me.');
6. },
7. (session, result, next) => {
8. session.dialogData.description = result.response;
9. var choices = ['high', 'normal', 'low'];
10. builder.Prompts.choice(session, 'Which is the severity of this problem?', choices);
11. },
12. (session, result, next) => {
13. session.dialogData.severity = result.response.entity;
14. builder.Prompts.text(session, 'Which would be the category for this ticket (software, hardware, networking, security or other)?');
15. },
16. (session, result, next) => {
17. session.dialogData.category = result.response;
18. var message = `Great! I'm going to create a \*\*${session.dialogData.severity}\*\* severity ticket in the \*\*${session.dialogData.category}\*\* category. ` +
19. `The description I will use is \_"${session.dialogData.description}"\_. Can you please confirm that this information is correct?`;
20. builder.Prompts.confirm(session, message);
21. },
22. (session, result, next) => {
23. if (result.response) {
24. session.send('Awesome! Your ticked has been created.');
25. session.endDialog();
26. } else {
27. session.endDialog('Ok. The ticket was not created. You can start again if you want.');
28. }
29. }

]);

**注:** Markdown 構文を使用して、よりリッチなテキスト メッセージを作成できることに注意してください。ただし、すべてのチャネルで Markdown がサポートされるわけではないので、注意することが重要です。

1. アプリを再実行して、エミュレーターの [Start new conversation] ボタン[](https://github.com/GeekTrainer/help-desk-bot-lab/blob/develop/Node/images/exercise2-start-new.png)を使用します。新しい会話をテストします。

[](https://github.com/GeekTrainer/help-desk-bot-lab/blob/develop/Node/images/exercise2-full-conversation-1.png)

この時点で再度ボットに話しかけると、ウォーターフォールが最初から開始されます。

**タスク 3: 外部 API を呼び出してチケットを保存**

この時点で、チケットのすべての情報が取得されましたが、この情報はウォーターフォールが終了すると破棄されます。今度は、外部 API を使用して、チケットを作成するコードを追加します。わかりやすくするため、チケットをインメモリ アレイに保存する単純なエンドポイントを使用します。実稼働環境では、ボットのコードからアクセスできる任意の API を使用できます。

**注:** ボットについての重要事項として、構築するほとんどのボットが既存の API のフロント エンドとなるということに留意してください。単純に言えば、ボットはアプリであり、人工知能 (AI)、機械学習 (ML)、または自然言語処理 (NLP) がなくてもボットとみなされます。

1. アプリのルート フォルダーで新しい **ticketsApi.js** ファイルを作成し、以下のコードを追加します。
2. var tickets = [];
3. var lastTicketId = 1;
4. module.exports = (req, res) => {
5. console.log('Ticket received: ', req.body);
6. let ticketId = lastTicketId++;
7. var ticket = req.body;
8. ticket.id = ticketId;
9. tickets.push(ticket);
10. res.send(ticketId.toString());

};

**タスク 4: サーバーを更新して API をホスト**

以下のステップでは、app.js のコードをクリーンアップして、サービスの追加のサポートを向上させます。さらに、Restify を更新して API と併用できるようにします。

1. app.js ファイルの冒頭に、以下の require ステートメントを追加します。

const ticketsApi = require('./ticketsApi');

1. listenPort 定数を追加します。

const listenPort = process.env.port || process.env.PORT || 3978;

1. ticketSubmissionUrl 定数を追加します。

const ticketSubmissionUrl = process.env.TICKET\_SUBMISSION\_URL || `http://localhost:${listenPort}`;

1. 以下に示すように、server.listen() を更新します。
2. server.listen(listenPort, '::', () => {
3. console.log('Server Up');

});

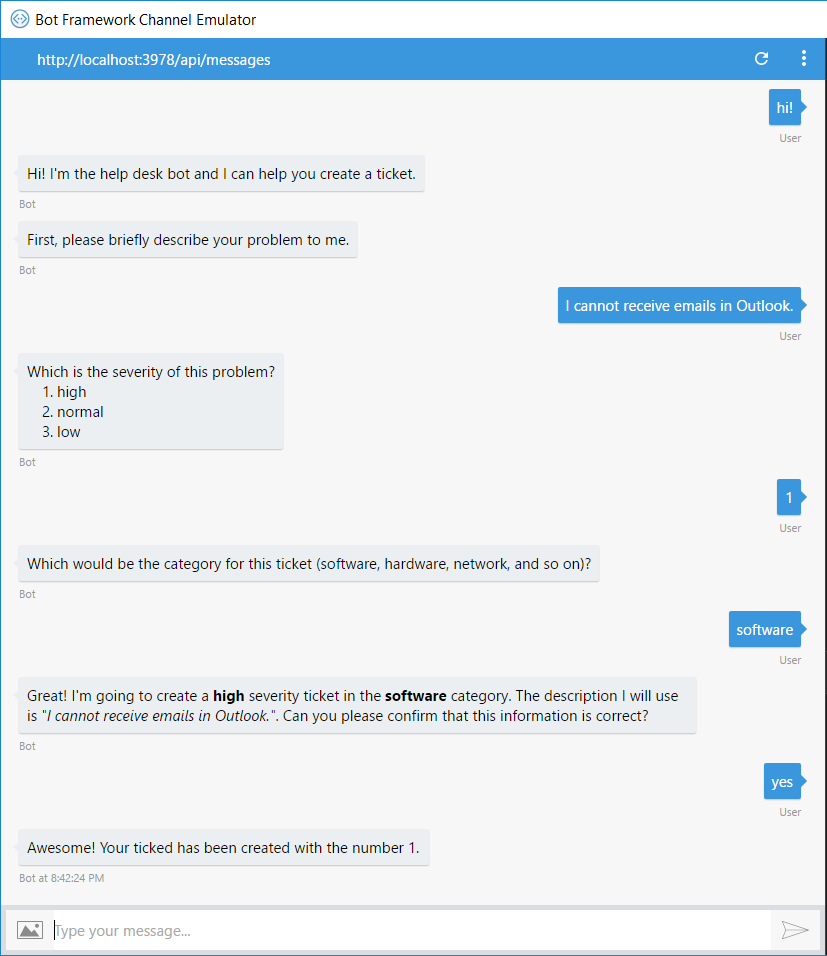
1. bodyParser を追加します。これにより、以下に示すように、API でメッセージ本文とチケット API を読み取れるようになります。
2. // Setup body parser and tickets api
3. server.use(restify.bodyParser());

server.post('/api/tickets', ticketsApi);

1. **最後のメッセージ ハンドラー**のコードを、以下のコードに置き換えます。このコードでは、dialogData をチケット API に送信します。
2. // --- existing code here ---
3. (session, result, next) => {
4. if (result.response) {
5. var data = {
6. category: session.dialogData.category,
7. severity: session.dialogData.severity,
8. description: session.dialogData.description,
9. }
10. const client = restify.createJsonClient({ url: ticketSubmissionUrl });
11. client.post('/api/tickets', data, (err, request, response, ticketId) => {
12. if (err || ticketId == -1) {
13. session.send('Something went wrong while I was saving your ticket. Please try again later.')
14. } else {
15. session.send(`Awesome! Your ticked has been created with the number ${ticketId}.`);
16. }
17. session.endDialog();
18. });
19. } else {
20. session.endDialog('Ok. The ticket was not created. You can start again if you want.');
21. }

}

1. ファイルを保存して、エミュレーターの [Start new conversation] ボタン[](https://github.com/GeekTrainer/help-desk-bot-lab/blob/develop/Node/images/exercise2-start-new.png)をクリックします。すべての会話を再度テストして、API からチケット ID が返されることを確認します。

[](https://github.com/GeekTrainer/help-desk-bot-lab/blob/develop/Node/images/exercise2-full-conversation-2.png)

**タスク 5: 通知メッセージを変更してアダプティブ カードを表示**

このタスクでは、チケットで[アダプティブ カード](http://adaptivecards.io/)を使用した後、ユーザーに表示される確認メッセージを向上させます。アダプティブ カードとは、開発者が一定の共通方式で UI コンテンツをやり取りできるようにするための、オープン ソースのカード交換フォーマットです。アダプティブ カードのコンテンツは、JSON オブジェクトとして指定できます。コンテンツはホスト アプリケーション (Bot Framework チャネル) 内でネイティブにレンダリングでき、ホストの外観に自動的に適合します。

カードを簡単に作成するため、カードの JSON は [ticket.json](https://github.com/GeekTrainer/help-desk-bot-lab/blob/develop/assets/exercise2-TicketSubmissionDialog/ticket.json) ファイルに既に用意されています。通常は、コンテンツに対して構造が既に作成されており、ランタイムに動的に追加します。ticket.json を探す際は、{ticketId}、{severity}、{category}、および {description} のプレースホルダーに注意します。オブジェクト内でこれらの文字列を探し、適切な値を指定して更新します。

1. アプリのルート フォルダーで、**cards** という名前のフォルダーを作成します。この新しいフォルダーで、このハンズオン ラボのルートにある [assets/exercise2](https://github.com/GeekTrainer/help-desk-bot-lab/blob/develop/assets/exercise2-TicketSubmissionDialog) フォルダーから **ticket.json** ファイルをコピーします。
2. 前のタスクで得られた app.js ファイルを開きます。require セクションで、読み取りを行う fs モジュールを以下のようにファイルに追加します。

const fs = require('fs');

1. ファイルの末尾に createCard 関数を追加します。この関数は、JSON ファイルのコンテンツを返し、プレースホルダーをパラメーターで置き換えます。
2. const createCard = (ticketId, data) => {
3. var cardTxt = fs.readFileSync('./cards/ticket.json', 'UTF-8');
4. cardTxt = cardTxt.replace(/{ticketId}/g, ticketId)
5. .replace(/{severity}/g, data.severity)
6. .replace(/{category}/g, data.category)
7. .replace(/{description}/g, data.description);
8. return JSON.parse(cardTxt);

};

1. ウォーターフォールの最後のステップを探し、次の行を置き換えます。

session.send(`Awesome! Your ticked has been created with the number ${ticketId}.`);

次の行で置き換えます。

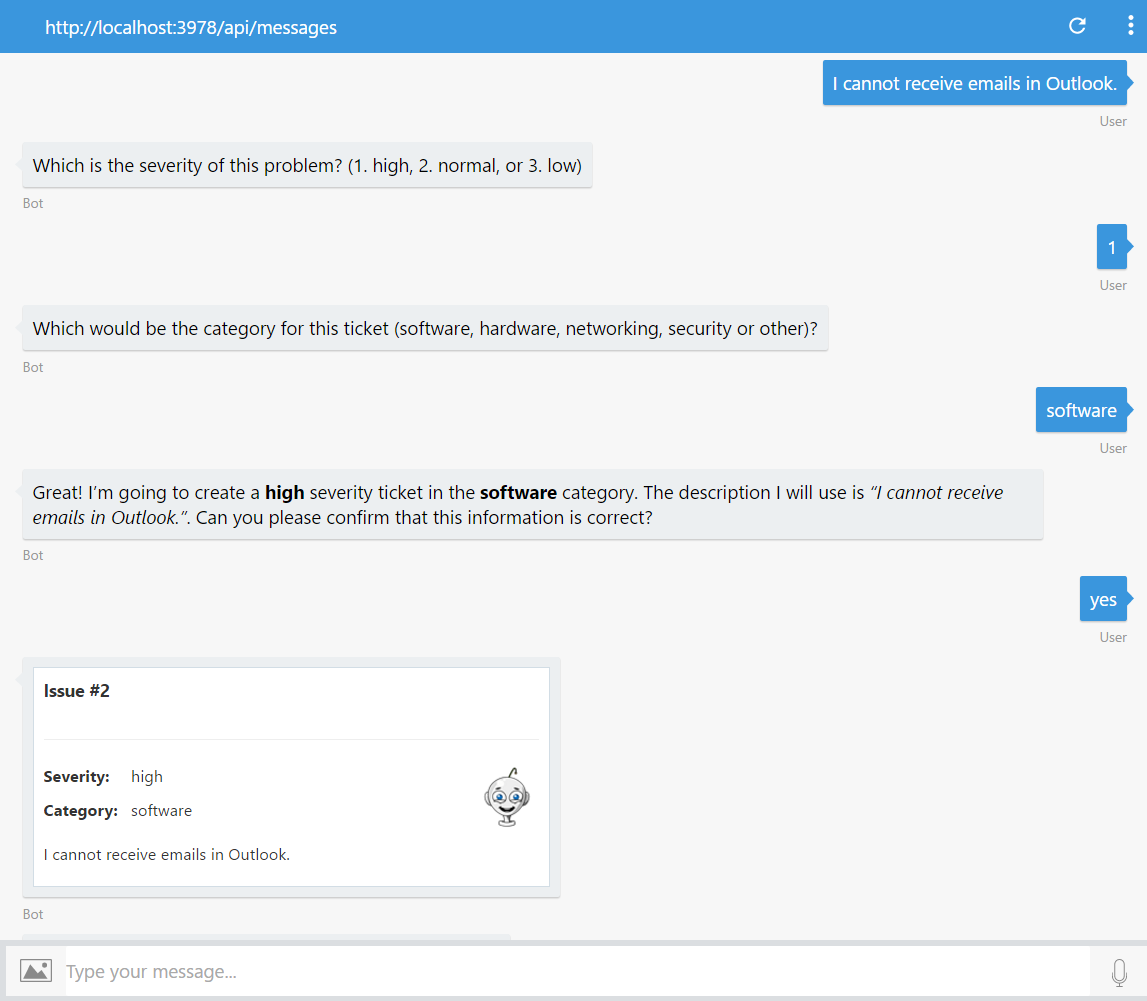
session.send(new builder.Message(session).addAttachment({

contentType: "application/vnd.microsoft.card.adaptive",

content: createCard(ticketId, data)

}));

1. ファイルを保存して、エミュレーターの [Start new conversation] ボタンを使用します[](https://github.com/GeekTrainer/help-desk-bot-lab/blob/develop/Node/images/exercise2-start-new.png)。新しい会話をテストします。確認メッセージについては、以下を参照してください。

[](https://github.com/GeekTrainer/help-desk-bot-lab/blob/develop/Node/images/exercise2-emulator-adaptivecards.png)

**その他の課題**

自主的に学習を続ける場合は、次のタスクを利用できます。

* conversationUpdate イベントを使用してボットにウェルカム メッセージを送信します。詳細は、[こちら](https://docs.microsoft.com/en-us/bot-framework/nodejs/bot-builder-nodejs-handle-conversation-events#greet-a-user-on-conversation-join)を参照してください。
* ボットがチケット API を呼び出す間、ボットにタイピング インジケーターを送信します。詳細は、[こちら](https://docs.microsoft.com/en-us/bot-framework/nodejs/bot-builder-nodejs-send-typing-indicator)を参照してください。