

Scheduling Bot

Meghna Pillai

Sometimes (well most of the time), humans tend to forget things. And while there may be several methods of scheduling reminders, having a chatbot is one of the most simple applications you can have to help with this. A scheduling chatbot can ask you several questions regarding the details of your event and puts it all together to send you a reminder one hour prior. In this tutorial, you will learn how to create a chatbot using Azure services that will make your life easier and up-to-date. There are three parts to creating this: developing the actual Bot, creating a web app, and creating an Azure function. Let's get started!

Downloading Programs

Let's start by downloading the following programs on your computer:

- Bot Framework Composer: [Windows](#) | [macOS](#) | [Linux](#)
- Bot Framework Emulator: <https://github.com/Microsoft/BotFramework-Emulator/releases>
- Postman: <https://www.postman.com/downloads/>
- SQL Server Management Studio: <https://aka.ms/ssmsfullsetup>
- Visual Studio: <https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&rel=16>

Creating An Account With Microsoft Azure

Once you have installed all five programs, create an account on Microsoft Azure if you have not already. If you have an account with Microsoft Azure, feel free to skip this step.

1. Go to <https://azure.microsoft.com/free>
2. Select the **Start free** option.
3. Sign in using either your Microsoft or GitHub account to create an account on Azure.
4. Once it redirects you to the **About you** page, choose your country/region. Then, enter your first and last name, email address, and phone number. Select **Next**.
5. On the **Identity verification by phone** page, choose your country code, and type in your phone number. Enter the verification code sent to you and select **Verify code**.
6. If the verification code is correct, it will prompt you to enter your credit card details. When done, select **Next**.
7. Review the agreement and privacy statement and select **Sign up**.

Creating New Folders To Store Files

When you are all signed up for Azure, the next step is to create a few folders on your computer.

1. Go to the local disk on your computer.
2. Create a folder. For our purposes, we will name the folder **Scheduling Bot**, but feel free to enter any name you prefer.

- Once you create a folder, create three other folders within the first one. Feel free to name it anything you like, but for our purposes, we will call it the following:

SchedulerBot (1)

SchedulerBotWebApp (2)

Scheduler Function (3)

Local Disk (C:) > Scheduling Bot >			Search Scheduling Bot
^	Name	Date modified	Type
	SchedulerBot (1)	7/28/2020 11:52 PM	File folder
	SchedulerBotWebApp (2)	7/28/2020 12:09 AM	File folder
	SchedulerFunction (3)	7/28/2020 12:09 AM	File folder

Composing The Bot (Stage 1)

We must now start composing the scheduler bot using the Bot Framework Composer.

*Side note: The Bot that we are creating will not have any advanced features, such as intents or Triggers, as it will only extend the `welcomeTheUser` activity. Regardless, it gets the main job done, and is a cool program to create!

Let's get started.

Phase 1 - Set Up

- Go to Bot Framework Composer (one of the five programs that you downloaded on your computer).
- Select `New` and then `Create from scratch` when you get to the home page.
- Under `Name`, give your Bot a name; it can be anything you would like. In this demo, we will call it `SchedulerBot`. And under `Location`, save your Bot to the first Bot folder that we made, which is inside the master folder that we also created. For instance, in this demo, the master folder is called `Scheduling Bot`, and we will be choosing our Location to be `SchedulerBot (1)`, the first of the three main folders inside the master. The other folders in the master are for the Web App and Function that we will be creating later on in this tutorial.

Define conversation objective

What can the user accomplish through this conversation? For example, BookATable, OrderACoffee etc.

Name *

SchedulerBot

Description

Location

C:\Scheduling Bot\SchedulerBot (1)

[create new folder](#)

4. When redirected, if you look to the left, under `Filter Dialog`, there should be a section that says `Greeting`. Select that.

Phase 2 - Building

Greeting Message

1. You should now be able to see the page in which we'll be making the functions for our Bot. Let's start by pressing the `+` sign, and then selecting the `Send a response` option (underneath `Loop: for each item` and `Branch: if/else`).
2. To the right, you will see a heading called `Language Generation` and a text box right beneath it. In this text box, we will enter a few potential greetings that the Bot will say to the user when they first sign on to the Bot. For this demo, you can type the following three statements (if you do choose to write your own, do not forget to put a dash and space before each one as typed below):

```
- Hello! I am your personalized scheduling bot. I can take note of appointments or events and remind you by text message one hour before.  
- Hi there! My name is Scheduling Bot. I can take note of appointments or events and remind you by text message one hour before.  
- Hey! I'm Scheduling Bot. I can take note of appointments or events and remind you by text message one hour before.
```

User Name

1. Now, the Bot needs to prompt the user for their name. To do this, click the `+` sign at the end of the page, hover over `Ask a question` and click on `Text`. To the right of your screen, you will see a heading that says, `Prompt for text`. In the text box below this, we will enter three ways in which the Bot can ask the user for their name. When in execution, the Bot will end up randomly picking one of these three options to use in the conversation each time. In this demo, we will put in the following:

```
- what is your name?  
- First, please give me your name.  
- Let's start with your name.
```

2. In the next step, the user must respond to the question that the Bot asked. So, above `Prompt for text`, you will see two tabs, `Bot Asks` and `User Input`. Select `User Input`. On this page, under `Property`, define the property as `user.name` for the purposes of this demo. By doing this, we are saving the value entered by the user for that question in a variable called `user.name`. In `Output format`, enter `=trim(this.value)`. We enter this because the command will remove any unnecessary spaces when the user enters in their name to the Bot.
3. Next to `User Input`, there should be an option called `Other`. In the text box underneath `Unrecognized prompt`, we must type a message that the user will receive if the max turn count (if mentioned) has exceeded and the default value is selected as the value. For this demo, please enter the following:

```
- Sorry I don't recognize ${this.value}. Please enter your name as text greater than 3 and less than 25 characters.
```

The `${this.value}` will allow the Bot to respond to the user using the response they entered in the first place.

4. Moving along, under `validation Rules`, we must create an expression that restricts the name that the user enters to be more than three characters and less than twenty-five and validates user input. Enter in the following to the right of `fx`:

```
=length(this.value) > 3 && length(this.value) < 25
```

Now, this will allow the Bot to only accept names that are greater than three and less than twenty-five characters.

*Important note: Whenever you enter a validation rule, make sure to hit enter at the end so that it registers. Otherwise, it will not save.

5. When you scroll down the side bar on the right, there will be a title called `Invalid prompt`. In this text box, we must type a message for the Bot to send when the user input does not meet any validation expression. Enter the following:

```
- Please enter a value between 3 and 25 characters.
```

User Phone Number

1. Once the user enters their name, the Bot now needs to find out their phone number so it can send text messages to the one we provide. Click the `+` sign at the bottom of the page, hover over `Ask a question`, and select `Text`. To the right, as before, we will see that under `Prompt for text`, there is an empty textbox. Let's enter in the following:

```
- Thanks ${user.name}. Now, what is your ten-digit US phone number?  
- Thank you ${user.name}. Now, please tell me your ten-digit US phone number.
```

These statements are two potential ways in which the Bot can ask the user for their number. The `${user.name}` will allow the Bot to respond to the user using whichever name they entered in the first place.

2. As said previously, the user must respond to the question that the Bot asked. Above `Prompt for text`, you will see two tabs, `Bot Asks`, and `User Input`. Select `User Input`. On this page, under `Property`, define the property as `user.phonenumber` for this demo. In `Output format`, enter `=trim(this.value)`. We enter this because the command will remove any unnecessary spaces when the user enters in their name to the Bot.
3. Next to `User Input`, there should be an option called `Other`. In the text box underneath `Unrecognized prompt`, we must type a message that the user will receive if the max turn count (if mentioned) has exceeded and the default value is selected as the value. For this demo, please enter the following:

```
- Please enter a valid ten-digit number.
```

4. When looking at `validation Rules`, we must create an expression that restricts the phone number that the user enters to be ten characters and validates user input. Enter in the following to the right of `fx`:

```
=length(this.value)==10
```

Now, this will allow the Bot to accept characters that are only equal to ten characters. For this demo, we are only validating the length of the input, and not the content.

5. When you scroll down the side bar on the right, there will be a title called `Invalid prompt`. In this text box, we must type a message for the Bot to send when the user input does not meet any validation expression. Enter the following:

```
- I don't recognized ${this.value} as a valid phone number.
```

The `${this.value}` will allow the Bot to respond to the user using the response they entered in the first place.

Bot Response to User Phone Number

1. Click the `+` sign at the end of the page and click on `Send a response`. In the text box below `Language Generation` on the right, enter the following:

```
- okay ${user.name}. I will send reminders to ${user.phonenumber}.
```

Using `${user.name}` will allow the Bot to send back (or echo) whichever name the user entered.

Using `${user.phonenumber}` will allow the Bot to send back (or echo) whichever phone number the user entered.

Event Date/Time

1. Now, the Bot needs to find out when the event is scheduled for. To do this, click the `+` sign at the end of the page, hover over `Ask a question` and click on `Date or time`. To the right of your screen, you will see a heading that says, `Prompt for text`. In the text box below this, we will enter three ways in which the Bot can ask the user for the event details. In this demo, we will put in the following:

```
- when should I schedule your appointment/event/reminder?
```

2. Select `User Input`. On this page, under `Property`, define the property as `user.appointmenttime`. Then go to `other`. In the text box beneath `Unrecognized prompt`, enter the following:

```
- Sorry, I don't recognize ${this.value}.
```

We won't need to enter any validation rules because when using date/time as the data type for a question, the Bot will automatically convert your answer into a date and a time.

When you scroll down the side bar on the right, there will be a title called `Invalid prompt`. In this text box, enter the following:

```
- Please tell me a time for the appointment.
```

Event Name

1. For now, our last step is to have the Bot ask the user what the name of the event or reminder is. Click the `+` sign at the end of the page, hover over `Ask a question` and click on `Text`. In the text box below `Prompt for text`, let's enter three ways in which the Bot can ask this question:

- what is the event?
- what is the title of the reminder?
- what would you like to call this event?

2. Select `user Input`. On this page, under `Property`, define the property as `user.event`.

We won't need to enter any validation rules.

Creating A SQL Database And Server On Azure

A SQL Server database is composed of a variety of tables that store specific sets of structured data. The tables contain rows and columns as you would see in any regular data table.


The purpose of creating a SQL database for this demo is so that you can store the information that the user enters into the Bot so that it can be retrieved later on, and also so that reminders can send out accordingly. Let's jump into it!

SQL Database

1. Go to Azure Portal using the following link: www.portal.azure.com. Log in to your account if you have not already.
2. On your home page, there should be an option called `Create a resource`. Once redirected to a new page, search for `SQL Database` and select the option that has the same name as the one you searched. Once you do so, hit `Create`.
3. Choose your subscription using the drop-down tool for `Subscription`. For `Resource group`, press Create new and provide it with a name of your choice. In this demo, we will be naming our resource group, `schedulingbot`.
4. When looking at the "Database details," give your database a name. We're going to call it `schedulingBotDB` in this demo.
5. For `Server`, select `Create a new server`. Give your server a name, and create an admin login and password as required. Make sure to take note of what your login and password are because we will need to use it later on in this tutorial. To simplify things, try not to include the `&` sign in your password as this will become an issue that needs additional steps to resolve later in the process. Next, choose the location which best correlates with yours. In the demo, we will use `(US) East US`.
6. Leave the question regarding the SQL elastic pool as `No`. Under `Compute + storage`, select the option that says `Configure database`. Once you redirect to a different page, press on `Basic` to the upper-left of your screen. And then, select the `Apply` option at the bottom of your screen.

Create SQL Database

Microsoft

 Changing Basic options may reset selections you have made. Review all options prior to creating the resource.

Basics Networking Additional settings Tags Review + create

Create a SQL database with your preferred configurations. Complete the Basics tab then go to Review + Create to provision with smart defaults, or visit each tab to customize. [Learn more](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ Azure subscription 1 ▼

Resource group * ⓘ (New) schedulingbot ▼

[Create new](#)

Database details

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

Database name * SchedulingBotDB ✓

Server * ⓘ (new) schedulingbot (East US) ▼

[Create new](#)

Want to use SQL elastic pool? * ⓘ ☐ Yes ☒ No

Compute + storage * ⓘ **Basic**
2 GB storage

[Review + create](#) [Next : Networking >](#)

7. Once you finish filling out this page, press **Review + create** and then **Create**. It may take a few minutes to deploy.
8. When your deployment is complete, go to the resource. You can do this in the following steps:
 - a. Go back to the Azure Portal home
 - b. Select **Resource groups**
 - c. Choose the name of your resource group that you entered in earlier (in the demo, it would be **schedulingbot**)
 - d. Choose the name of your database at the bottom of your resource group's overview page (in the demo, it would be **SchedulingBotDB**)
9. In the overview page of your database, you'll be able to find a heading that says, **Connection strings**. Click on the link below that (**Show database connection strings**), and it should redirect you to a different page. Once you're there, make sure to take note of the connection string in the text box, and copy it somewhere as we will need it later on.
10. Once you do copy down the connection string, go back to it (the copied version). Somewhere along that line, you will see **Password={your_password}**. What you need to do is copy your password for your database and replace it with **{your_password}** in this string. Once again, make sure to save the entire connection string because you will need it soon.

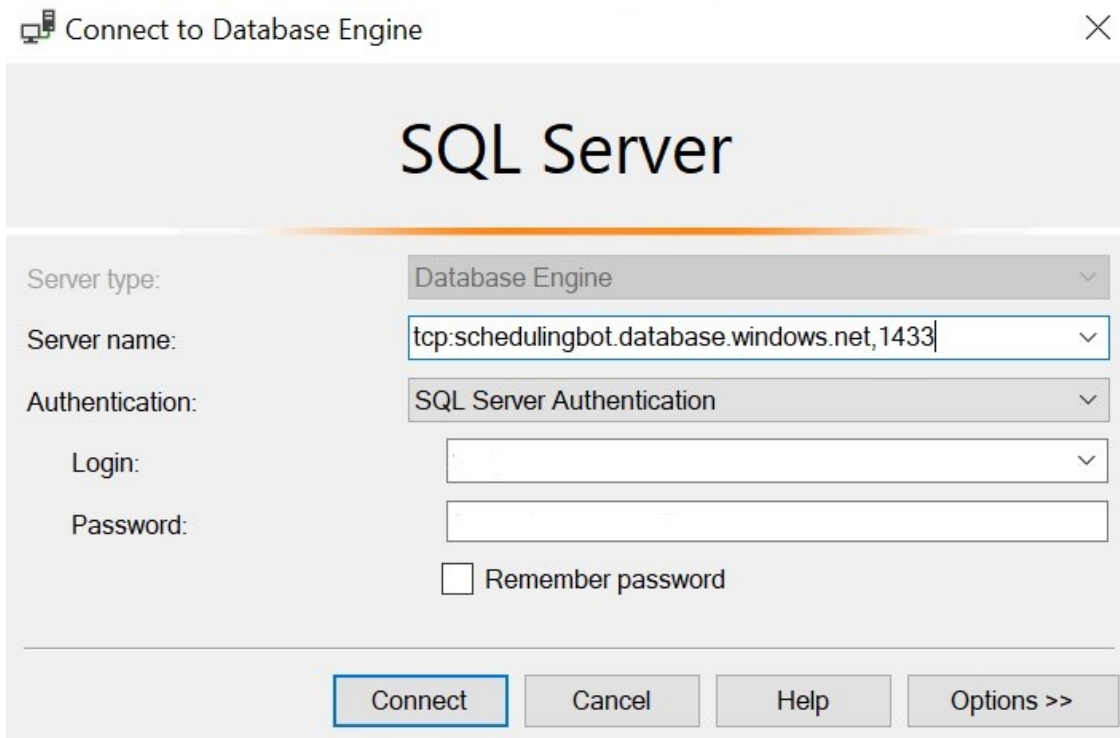
SQL Server

1. Go back to your resource group, and instead of choosing the name of your database, select your SQL server. For the demo, it's label would be `schedulingbot` as well. When redirected, you'll be able to find a heading that says, `Firewalls and virtual networks`. Click on the link below (`show firewall settings`).
2. When you are on the `Firewalls and virtual networks` page, if you scroll a bit down, you will see a title that says, `Allow Azure services and resources to access this server`. Change the `No` to a `Yes`.
3. Immediately after, you will be able to see your Client IP address - copy this number. Scrolling along with the page, there's a small table with three key components: `Rule name`, `Start IP` and `End IP`. For `Rule name`, put in the following for this demo: `Access From Home` (as a note, you do not have to put in "Home," you can enter the place as wherever you would like to access it from!). Next, for `Start IP` and `End IP`, paste the Client IP address that you previously copied. When finished, make sure to hit save at the top of your page.

Setting Up The Database

The purpose of setting up the database is so that we can create a table to hold the event details that the Bot collects from users. Let's get into it!

1. Open Microsoft SQL Server Management Studio (a program that you downloaded at the beginning of this tutorial). The following screen should pop up:



2. Using the connection string that you noted down before, extract the server name, login, and password. Enter those values into this pop-up. Also, change your `Authentication` to `SQL Server Authentication`.
3. Once you are logged in to your database, press `Ctrl + N` to open a new query. Enter the following code in your new query:

```
if(OBJECT_ID('Schedule', 'U') is not null)
drop table dbo.Schedule;

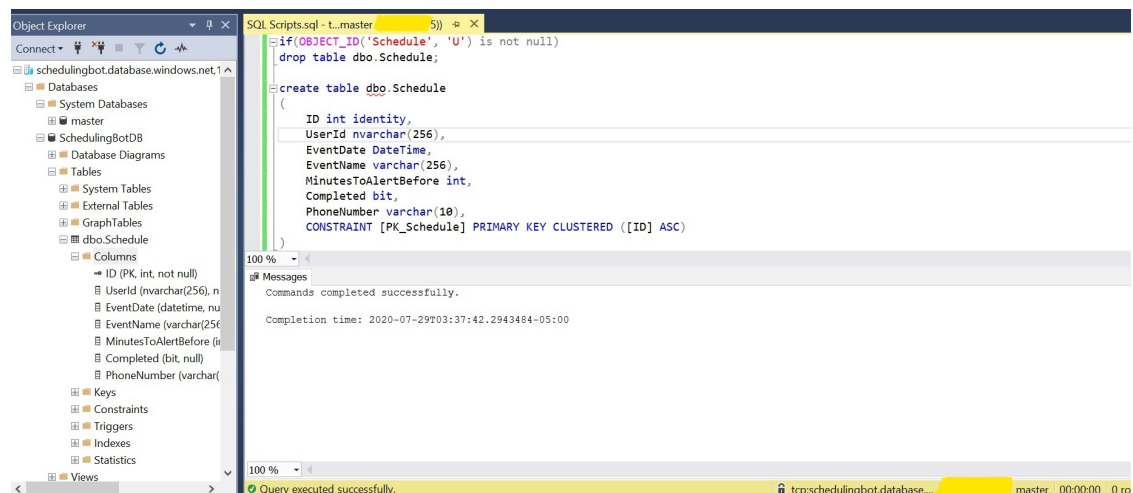
create table dbo.Schedule
```



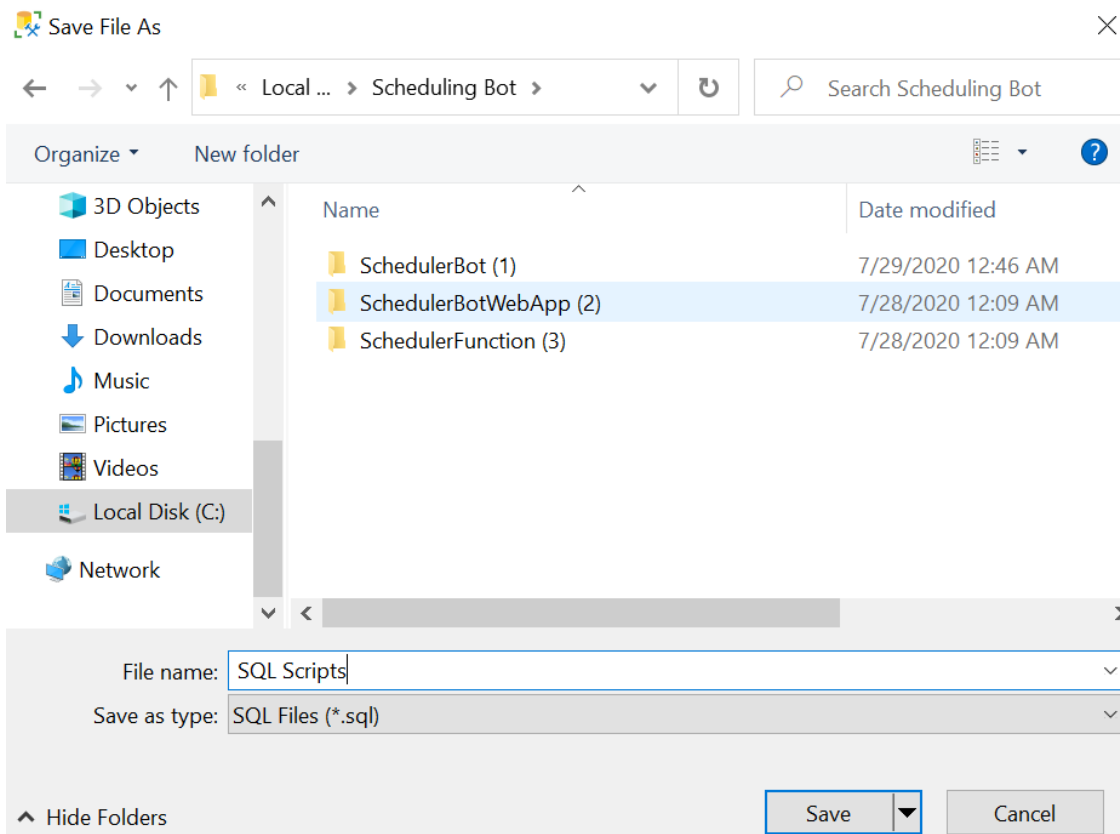
```
(
    ID int identity,
    UserId nvarchar(256),
    EventDate DateTime,
    EventName varchar(256),
    MinutesToAlertBefore int,
    Completed bit,
    PhoneNumber varchar(10),
    CONSTRAINT [PK_Schedule] PRIMARY KEY CLUSTERED ([ID] ASC)
)
```

When describing the code, the first two lines delete the table if it already exists, and the rest of the lines create the table that we need to store user input.

4. Change the database at the top of the screen from **Master** to the one you created. Otherwise, the table will end up in the Master database instead of the one created for this Bot. In this demo, the database we created is labeled as **SchedulingBotDB**.
5. Select all of the code and press **Execute**. What this code does is create a new table for you to store all the data that comes from the Bot when a user enters in the details regarding a reminder or event.
6. Press the blue refresh button towards the upper left of your screen. Once you refresh, you should be able to see that you created a table with several columns to store your all the relevant data from your Bot. It should look similar to this:



7. Save your work to the **Scheduling Bot** folder (the master folder created at the beginning of this tutorial, which already has three sub-folders).



8. Press **Ctrl + N** to open a new query. Enter the following code in your new query:

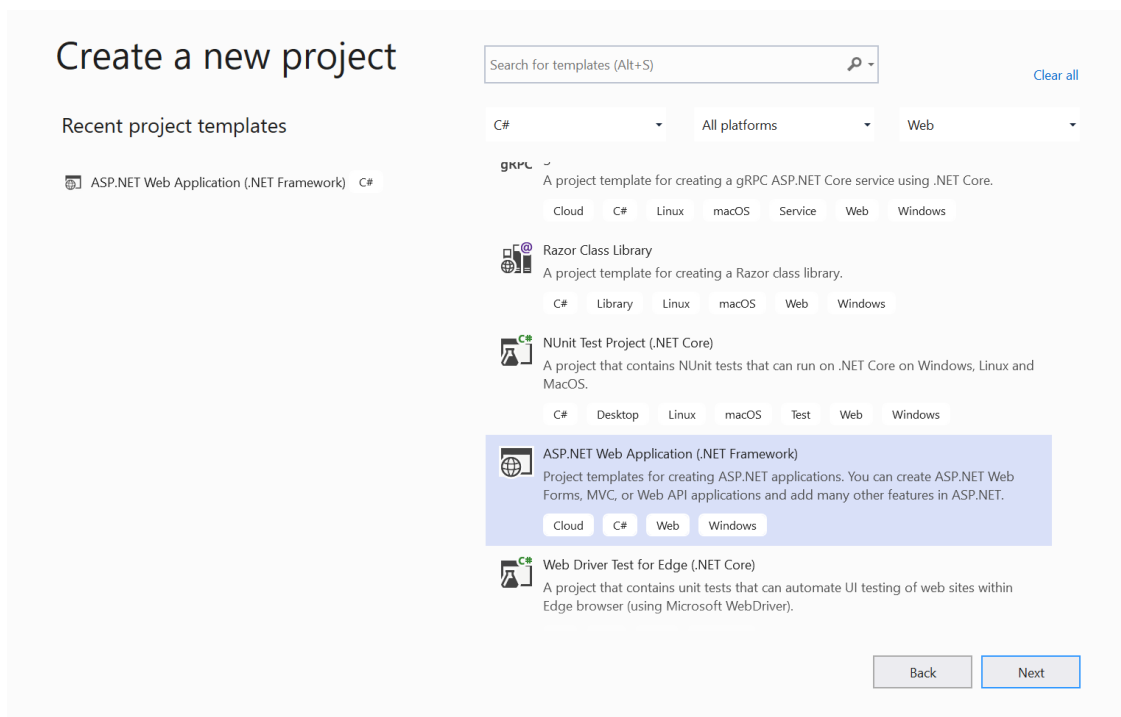
```
select *from dbo.schedule
```

Select the code and hit **Execute**. When you do, you should see an empty table with the following columns: **ID**, **UserId**, **EventDate**, **EventName**, **MinutesToAlertBefore**, **Completed**, **PhoneNumber**.

Setting Up The Scheduling Bot Web App (And APIs)

Creating A New Project

1. Now, let's go onto Visual Studio (another application that you installed during the beginning of this tutorial). Once it's opened up, select **Create a new project**. It should redirect you to a new page. There are several things we have to do here:
 - a. Select your language as **C#**.
 - b. Change your project type to **web**.
 - c. Choose **ASP.NET web Application (.NET Framework)**
 - d. Hit **Next**.

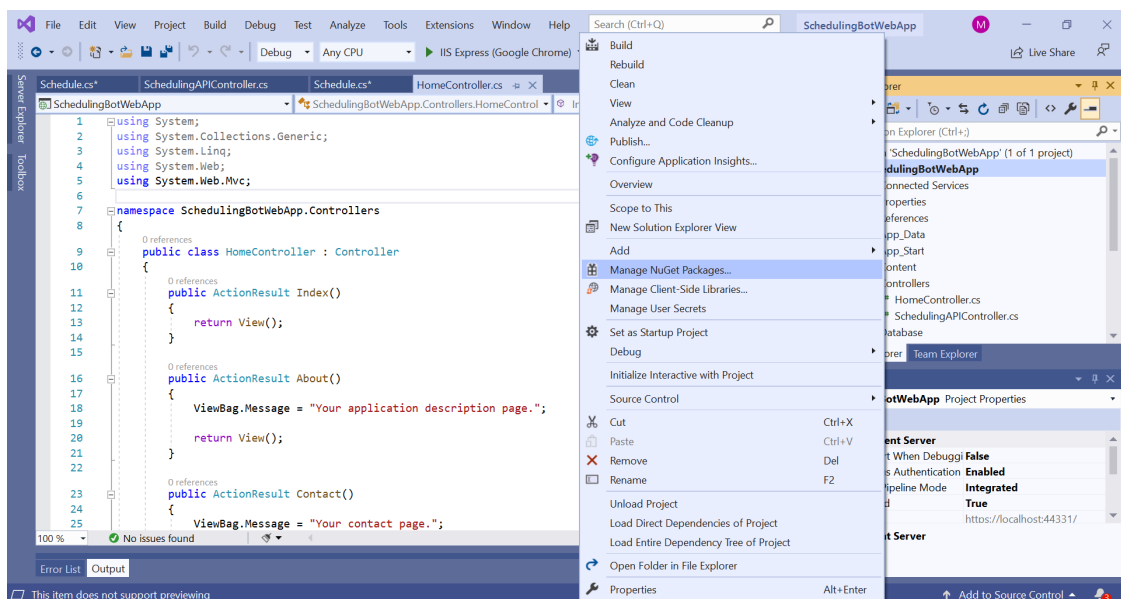


- Now, you must name the project. In this demo, we're going to name it `SchedulingBotWebApp`. Once named, make sure to save it to the Web App folder in your master folder (The names of the folders may be different depending on what you inputted originally). If you named the folders according to what the demo described, then you should save it to the `SchedulerBotWebApp (2)` folder in `Scheduling Bot`.

Installing Dapper

Before we get started, we need to download a NuGet package necessary for this particular project. We will use `Dapper`, the NuGet package we that we are installing, to get the API to talk to the database, and also to convert the data retrieved from the database into a corresponding data model defined in the API.

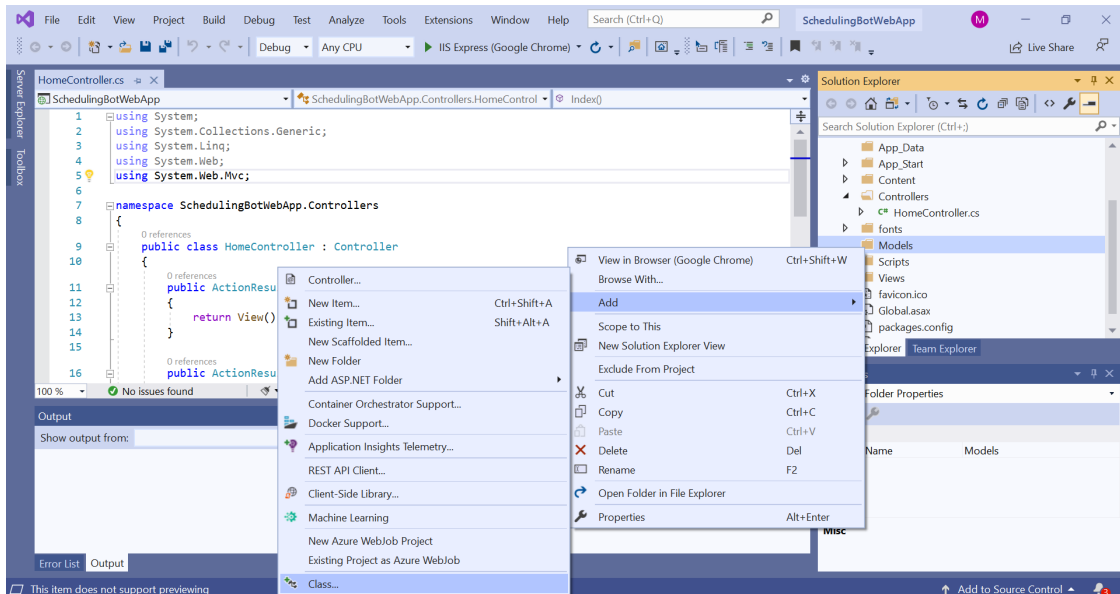
- To do this, right click on `SchedulingBotWebApp` and click on `Manage NuGet Packages...`. Here's a visual to help:



- When redirected, select `Browse` and look up `Dapper`. Then click on it and follow the procedures for installation.

Creating a New Class in Models

1. If you look to the right, you should see a **Solution Explorer**. Scroll down to find the **Models** folder, and right click on it. Then, hover over **Add** and select **Class...**.

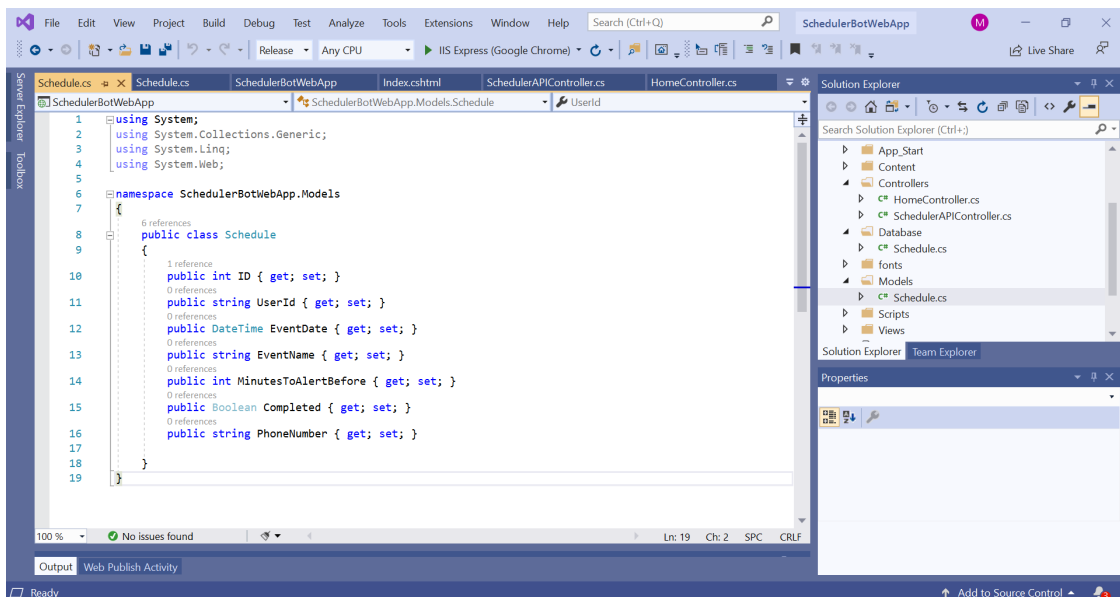


Rename the class as **Schedule** for now.

2. When redirected to the new class you added, insert the following code into **public class Schedule {}**:

```
public int ID { get; set; }
public string UserId { get; set; }
public DateTime EventDate { get; set; }
public string EventName { get; set; }
public int MinutesToAlertBefore { get; set; }
public Boolean Completed { get; set; }
public string PhoneNumber { get; set; }
```

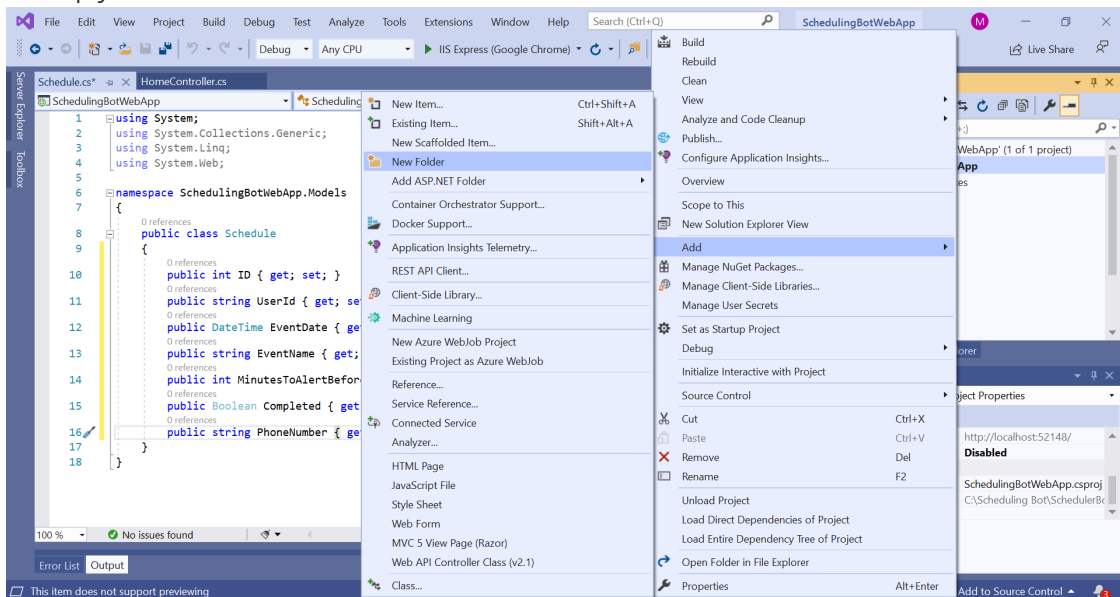
It should look like this:



The code you just entered is the data structure of the table. It matches with what's in the database.

Creating a New Class in Database

1. Let's create a new folder. In your **Solution Explorer**, right click on **SchedulingBotWebApp**. Then, hover over **Add** and select **New Folder**. Name the folder **Database**. Here's a visual to help you do this:



2. Now, as we did before, we need to create a new class in this folder. To do so, scroll down to find the **Database** folder, and right click on it. Then, hover over **Add** and select **Class...**. Rename the class as **Schedule**.
3. When redirected to the new class you added, insert the following code into **public class schedule {}**:

```
{  
    private string ScheduleAdd = "insert into dbo.Schedule (UserId,  
    EventDate, EventName, MinutesToAlertBefore, Completed, PhoneNumber) values  
    (@UserId, @EventDate, @EventName, @MinutesToAlertBefore, 0, @PhoneNumber);  
    select SCOPE_IDENTITY()";  
    private string SchedulesGet = "select * from dbo.Schedule where  
    Completed = 0 and eventdate < DATEADD(minute, minutestoalertbefore,  
    DATEADD(hour, 5, getdate()))";  
    private string ScheduleComplete = "update dbo.Schedule set Completed  
    = 1 where ID = @ID";  
  
    public string ConnectionString =  
    ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString  
    ;  
    public int AddSchedule(SchedulerBotWebApp.Models.Schedule schedule)  
    {  
        using (IDbConnection db = new SqlConnection(ConnectionString))  
        {  
            return db.Execute(ScheduleAdd, schedule, CommandType:  
            CommandType.Text);  
        }  
    }  
  
    public List<SchedulerBotWebApp.Models.Schedule> GetSchedules()  
    {  
        using (IDbConnection db = new SqlConnection(ConnectionString))  
        {  
            return db.Query<SchedulerBotWebApp.Models.Schedule>  
            (SchedulesGet, CommandType: CommandType.Text).ToList();  
        }  
    }  
}
```

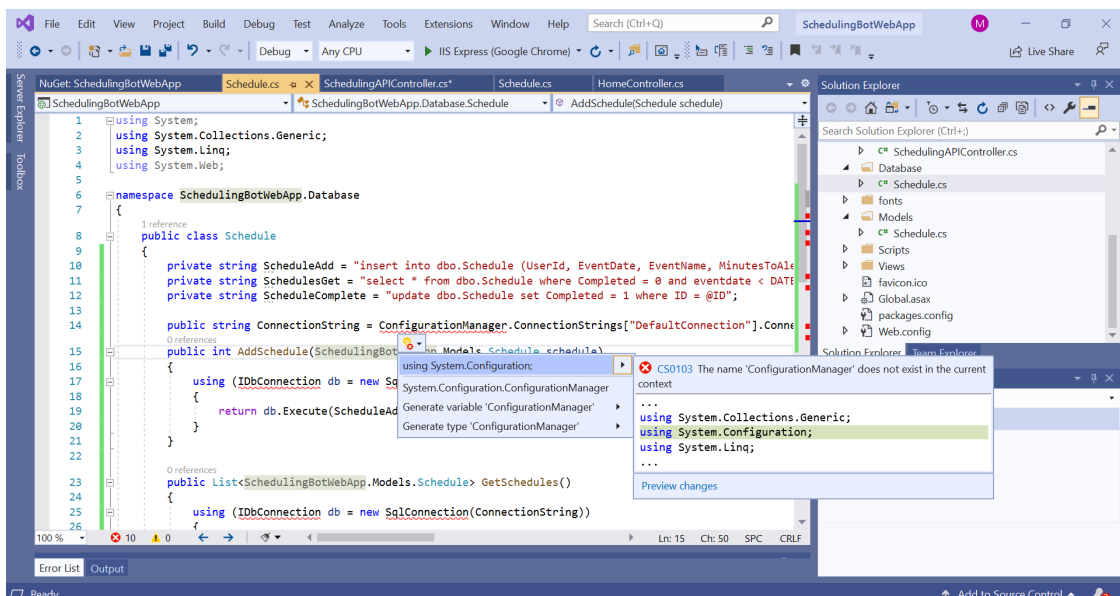
```

    }

    public void
    MarksScheduleAsCompleted(SchedulerBotWebApp.Models.Schedule schedule)
    {
        using (IDbConnection db = new SqlConnection(ConnectionString))
        {
            db.Execute(ScheduleComplete, new { Id = schedule.ID },
            commandType: CommandType.Text);
        }
    }
}

```

4. Once you deploy the code, you may realize that a few lines have been underlined in red, indicating some sort of issue. Hover over each line with red underlined, press the light bulb symbol, and select the first suggestion that pops up to resolve the issues. Here's a visual to guide you:



5. Once you are done, there should no longer be any visible errors on your page.

What you just created is your data layer. It allows you to send data (by inserting a row into the table), read data, and update data in the database. We have created three methods to match those actions accordingly in the code you inserted. Your controller will end up calling this class that you created. To describe what a controller is, it is the front-end. Your Bot and your Azure Function, which we have not created yet, is what we will end up exposing your controller to.

Creating a New Class in Controllers

1. Scroll to find the `Controllers` folder, and right click on it. Then, hover over `Add` and select `Class...`. Name the class `SchedulerApiController`.
2. When redirected to the new class you added, replace `public class Schedule {}` with the following code:

```

public class SchedulerApiController : ApiController
{
    SchedulerBotWebApp.Database.Schedule _schedule = new
    Database.Schedule();
}

```

```

[Route("SaveSchedule")]
[HttpPost]
public IActionResult SaveSchedule([FromBody] Schedule schedule)
{
    try
    {
        var id = _schedule.AddSchedule(schedule);
        return Ok(new { Status = "Ok" });
    }
    catch (Exception ex)
    {
        return InternalServerError(ex);
    }
}

[Route("GetSchedules")]
[HttpGet]
public IActionResult GetSchedules()
{
    try
    {
        var schedules = _schedule.GetSchedules();
        return Ok(schedules);
    }
    catch (Exception ex)
    {
        return InternalServerError(ex);
    }
}

[Route("CompleteSchedule")]
[HttpPost]
public IActionResult CompleteSchedules([FromBody] Schedule
schedule)
{
    try
    {
        _schedule.MarkScheduleAsCompleted(schedule);
        return Ok();
    }
    catch (Exception ex)
    {
        return InternalServerError(ex);
    }
}
}

```

3. As said before, once you deploy the code, you may realize that a few lines have been underlined in red, indicating some sort of issue. Hover over each line with red underlined, press the light bulb symbol, and select the first suggestion that pops up to resolve the issues.

The class we just created has three main functionalities as well: it saves the schedule, it gets schedules, and it also marks them as complete. Here you are calling the `schedule.cs` class that you created within the database folder, which in turn calls the database using a Connection String.

Editing web.config

1. Go back to your **Solution Explorer**. Scroll down to find **web.config** and click on it.
2. Immediately beneath **</appSettings>**, enter the following code. Remember to replace **{your_connection_string}** with your own connection string that you took note of towards the beginning of this tutorial.

```
<connectionStrings>
  <add name="DefaultConnection" connectionString="
{your_connection_string}" providerName="System.Data.SqlClient" />
</connectionStrings>
```

Finally, remember to save all of your work by clicking **File** and then **Save all**.

Publishing Web App To Azure

1. Now it's time for us to publish everything to Azure. To do this, right click on **SchedulingBotWebApp**, and select **Publish...**. When it asks, **where are you publishing today?**, choose **Azure**. And then, when it asks, **which Azure service would you like to use to host your application?**, select **Azure App Service (Windows)**.
2. At the bottom, you should see **Create a new Azure App Service...**. Select that. Fill out the following groups as shown below, and press **Create** when done:

Name	<input type="text" value="SchedulingBotWebApp"/>
Subscription	<input type="text" value="Azure subscription 1"/>
Resource group	<input type="text" value="schedulingbot (East US)"/> New...
Hosting Plan	<input type="text" value="SchedulerBotWebApp (East US, F1)"/> New...

When the process is complete, your Azure Web App Bot should now be deployed. When you hit **Finish** and close out of the tab, also remember to select **Publish** to finalize the deployment.

Once this is done, you can utilize Postman, a software development tool that allows people to test calls to APIs, as a troubleshooting tool if anything goes wrong. In other words, if something does not work, you can eliminate the API as the problem using Postman.

Composing the Bot (Stage 2)

Sending An HTTP Request

1. We must now finish composing the Bot using the Bot Framework Composer. When you go back, scroll to the bottom of your screen and press the **+** symbol. Hover over **Access external resources** and select **Send an HTTP request**.
2. On the right, change your **HTTP Method** to **POST**.
3. For your URL, go back to Azure Portal and find your Web App that you deployed via Visual Studio (it should be in your resource group). Once you select it, in the **overview** page, you should be able to see the URL. Copy it and paste it to the URL slot in Bot Framework Composer. Then, at the end, add this: `/saveSchedule`. So, in this demo, the final URL would be: <https://schedulingbotwebapp.azurewebsites.net/SaveSchedule>
4. In the text box underneath **Body**, enter the following:

```
{
  "UserId": "${user.name}",
  "EventDate": "${user.appointmenttime[0].value}",
  "EventName": "${user.event}",
  "MinutesToAlertBefore": 60,
  "PhoneNumber": "${user.phonenumber}"
}
```

5. For **Result property**, enter the following: `dialog.api_response`

Branch: If/else

1. Press the **+** symbol. Hover over **Create a condition** and select **Branch: If/else**.
2. Set the condition as: `dialog.api_response.statusCode == 200`

A Status Code of 200 is what the API returns when everything goes well. Anything other than a 200 denotes that something went wrong.

3. For when the condition is true, underneath **True**, press the **+** symbol and click **Send a response**. Under **Language Generation**, enter the following:

```
- ok. I will remind you at ${user.appointmenttime[0].value} to
  ${user.event}. Goodbye.
```

4. For when the condition is false, underneath **False**, press the **+** symbol and click **Send a response**. Under **Language Generation**, enter the following:

```
- I got an error: ${dialog.api_response.content}
```

Delete Properties

1. Scroll to the bottom of your screen and press the **+** symbol. Hover over **Manage properties** and select **Delete properties**.
2. Enter the following properties as described in the image:

Properties * ?

string ▾

user.name

string ▾

user.phonenumber

string ▾

user.appointmenttime

string ▾

user.event

string ▾

dialog.api_response

Test Bot

1. In the upper right corner, select **Start Bot**. Once that loads, press **Test in Emulator**.
2. When redirected to the Bot Framework Emulator, test your bot to make sure it does not display any errors.

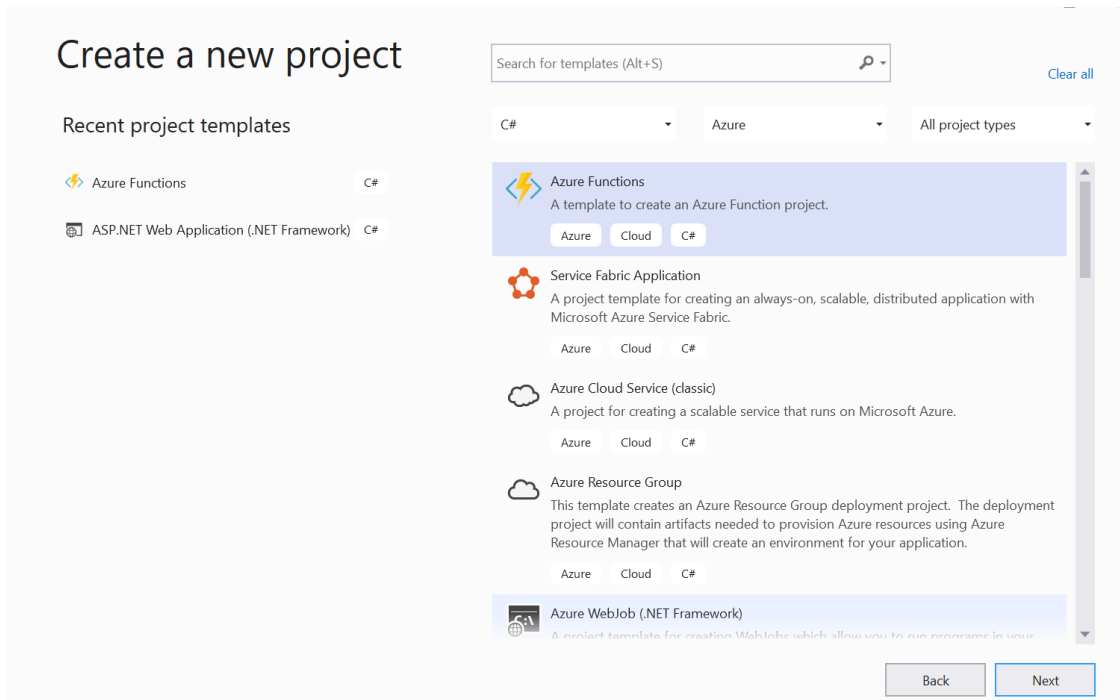
Setting Up Twilio

1. Go to <https://www.twilio.com/> and **Sign up**.
2. Enter in your **First Name**, **Last Name**, **Email**, and **Password** to create an account. Hit **Start your free trial** and verify your email.
3. Verify your phone number. Then, answer the series of questions that are presented to you.
4. Once you get to the home page, select **Create a Trial Number**. Copy down your new trial phone number, your **Account SID**, and your **Auth Token** as we will be needing it later for the Azure Function.

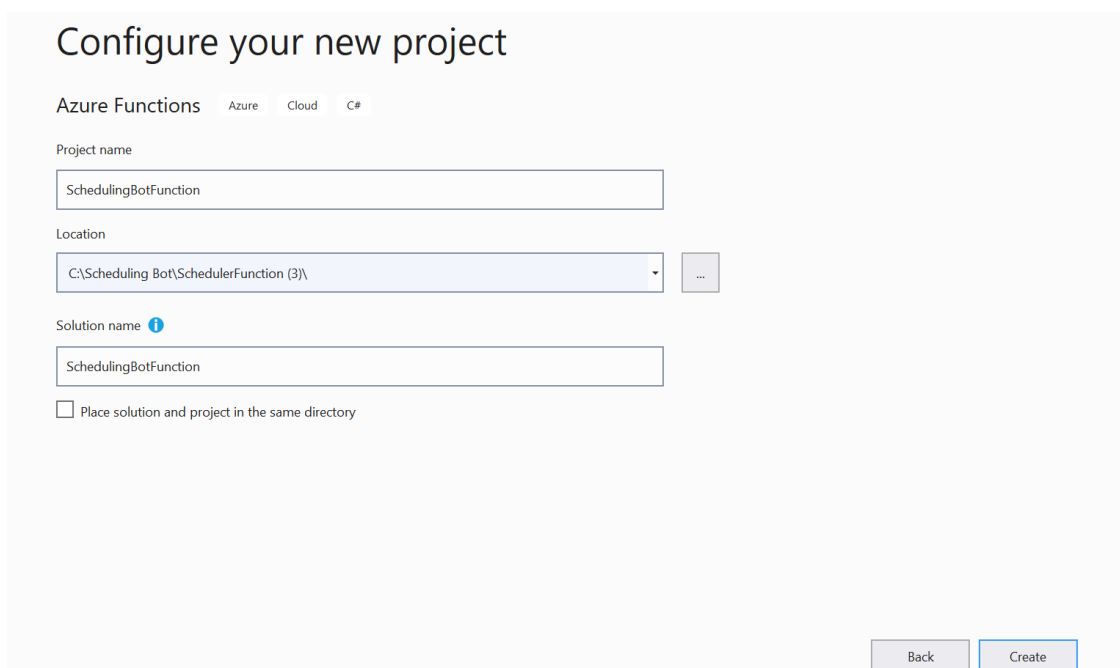
Creating An Azure Function

Creating A New Project

1. Now, let's go onto Visual Studio. Once it's opened up, select **Create a new project**. It should redirect you to a new page. There are several things we have to do here:
 - a. Select your language as **C#**.
 - b. Change your platform to **Azure**. Change your project type to **web**.
 - c. Choose **Azure Functions**.
 - d. Hit **Next**.



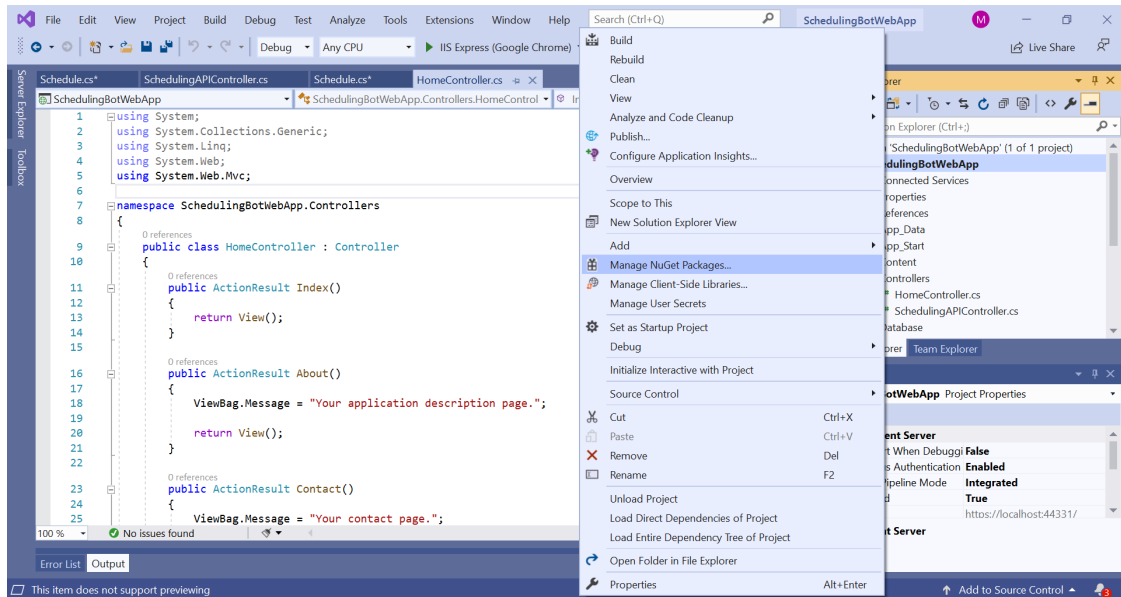
2. Now, you must name the project. In this demo, we're going to name it **SchedulingBotFunction**. Once named, make sure to save it to the Function in your master folder (The names of the folders may be different depending on what you inputted originally). If you named the folders according to what the demo described, then you should save it to the **SchedulerFunction (3)** folder in **Scheduling Bot**.



3. Select **Timer Trigger**. Change the 5 (runs every 5 minutes) to a 1 (runs every one minute), and press **Create**.

Installing Twilio

1. Before we get started, we need to download a NuGet package necessary for this particular project. To do this, right click on `SchedulingBotWebApp` and click on `Manage NuGet Packages...`. Here's a visual to help:



2. When redirected, select `Browse` and look up `Twilio`. Then click on it and follow the procedures for installation.

Set Up

1. Enter the following code immediately above the line that says `log.LogInformation`. Remember to replace certain parts of your code, as listed, with your Get URL, Account SID, Auth Token, Twilio phone number, and your Update URL. For your Get URL, check Azure for your web app URL, and add `/Getschedules` to the end of it. For your Update URL, use your web app URL, and add `/CompleteSchedule` to the end of it.

```
using (var httpClient = new HttpClient())
{
    string geturl = $"{your_get_url}";
    var response = await httpClient.GetAsync(new Uri(geturl));
    List<Schedule> schedules = new List<Schedule>();

    if (response.IsSuccessStatusCode)
    {
        schedules =
            JsonConvert.DeserializeObject<List<Schedule>>(await
                response.Content.ReadAsStringAsync());
    }

    foreach (var schedule in schedules)
    {
        // Your Account SID from twilio.com/console
        var accountSid = "{your_account_SID}";
        // Your Auth Token from twilio.com/console
        var authToken = "{your_AUTH_token}";

        TwilioClient.Init(accountSid, authToken);

        var message = MessageResource.Create(
            to: new PhoneNumber("schedule.PhoneNumber"),
            from: new PhoneNumber("{your_Twilio_phone_number}"),
```

```

        body: $"Reminder from SchedulerBot.
{schedule.EventName} at {schedule.EventDate}");

        string updateurl = $"{{your_update_URL}}";
        var content = new
StringContent(JsonConvert.SerializeObject(schedule), Encoding.UTF8,
"application/json");

        using (var _httpClient = new HttpClient())
        {
            var r = await _httpClient.PostAsync(updateurl,
content);
        }
    }
}

```

2. Once you deploy the code, you may realize that a few lines have been underlined in red, indicating some sort of issue. Hover over each line with red underlined, press the light bulb symbol, and select the first suggestion that pops up to resolve the issues.

Publishing Function To Azure

1. Now it's time for us to publish everything to Azure. To do this, right click on `SchedulingBotFunction`, and select `Publish...`. When it asks, `where are you publishing today?`, choose `Azure`. And then, when it asks, `which Azure service would you like to use to host your application?`, select `Azure Function App (windows)`.
2. At the bottom, you should see `Create a new Azure Function...`. Select that. Fill out the following groups as shown below, and press `Create` when done:

Name	<input type="text" value="SchedulingBotFunction"/>
Subscription	<input type="text" value="Azure subscription 1"/>
Resource group	<input type="text" value="schedulingbot (East US)"/> New...
Plan Type	<input type="text" value="Consumption"/>
Location	<input type="text" value="East US"/>
Azure Storage	<input type="text" value="schedulerfunction2020072 (East US)"/> New...

When the process is complete, your Azure Function should now be deployed. When you hit **Finish** and close out of the tab, also remember to select **Publish** to finalize the deployment.

Test Your Bot

Go back to Bot Framework Composer, restart your Bot, and test it in the Emulator. If you go through it, and all goes well, you should receive a text message depending on what time your event is (the Bot will remind you one hour prior), and your Bot should be fully functioning. If not, remember to go to Postman or your Application Insights to help figure out the errors or issues in your program. You will have to use Application Insights for either the API, the Azure Function, or both depending on where the error is. Enjoy coding!