### Install Postman

This tutorial uses Postman to test the web API.

* Install [Postman](https://www.getpostman.com/downloads/)
* Start the web app.
* Start Postman.
* Disable **SSL certificate verification**
  + From **File** > **Settings** (**General** tab), disable **SSL certificate verification**.

**Warning**

Re-enable SSL certificate verification after testing the controller.

### Test PostTodoItem with Postman

* Create a new request.
* Set the HTTP method to POST.
* Set the URI to https://localhost:<port>/api/todoitems. For example, https://localhost:5001/api/todoitems.
* Select the **Body** tab.
* Select the **raw** radio button.
* Set the type to **JSON (application/json)**.
* In the request body enter JSON for a to-do item:

JSONCopy

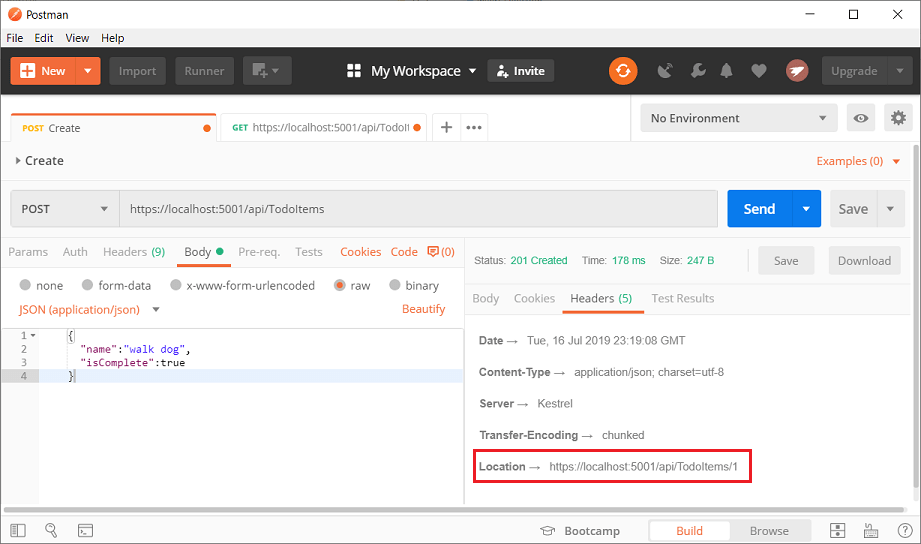
{

"name":"walk dog",

"isComplete":true

}

* Select **Send**.

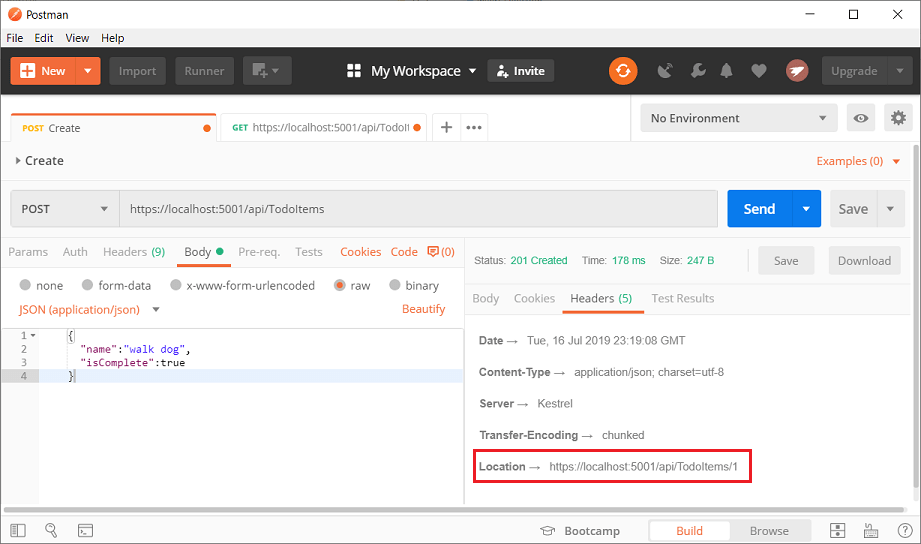


### Test the location header URI

The location header URI can be tested in the browser. Copy and paste the location header URI into the browser.

To test in Postman:

* Select the **Headers** tab in the **Response** pane.
* Copy the **Location** header value:



* Set the HTTP method to GET.
* Set the URI to https://localhost:<port>/api/todoitems/1. For example, https://localhost:5001/api/todoitems/1.
* Select **Send**.

## Examine the GET methods

Two GET endpoints are implemented:

* GET /api/todoitems
* GET /api/todoitems/{id}

Test the app by calling the two endpoints from a browser or Postman. For example:

* https://localhost:5001/api/todoitems
* https://localhost:5001/api/todoitems/1

A response similar to the following is produced by the call to GetTodoItems:

JSONCopy

[

{

"id": 1,

"name": "Item1",

"isComplete": false

}

]

### Test Get with Postman

* Create a new request.
* Set the HTTP method to **GET**.
* Set the request URI to https://localhost:<port>/api/todoitems. For example, https://localhost:5001/api/todoitems.
* Set **Two pane view** in Postman.
* Select **Send**.

This app uses an in-memory database. If the app is stopped and started, the preceding GET request will not return any data. If no data is returned, [POST](https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-5.0&tabs=visual-studio#post) data to the app.

### Test the PutTodoItem method

This sample uses an in-memory database that must be initialized each time the app is started. There must be an item in the database before you make a PUT call. Call GET to ensure there's an item in the database before making a PUT call.

Update the to-do item that has Id = 1 and set its name to "feed fish":

JSONCopy

{

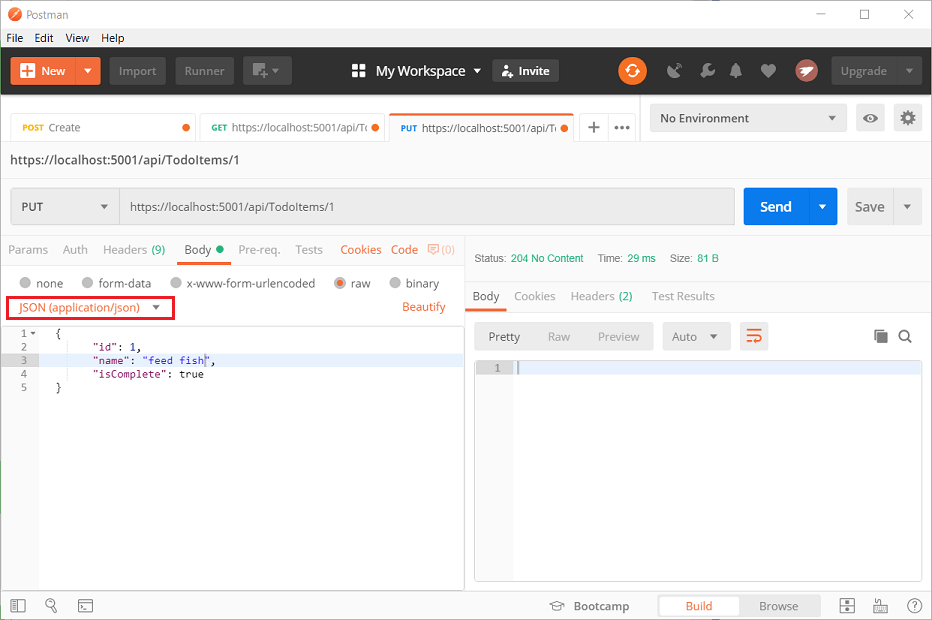
"Id":1,

"name":"feed fish",

"isComplete":true

}

The following image shows the Postman update:



## The DeleteTodoItem method

Examine the DeleteTodoItem method:

// DELETE: api/TodoItems/5

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteTodoItem(long id)

{

var todoItem = await \_context.TodoItems.FindAsync(id);

if (todoItem == null)

{

return NotFound();

}

\_context.TodoItems.Remove(todoItem);

await \_context.SaveChangesAsync();

return NoContent();

}

### Test the DeleteTodoItem method

Use Postman to delete a to-do item:

* Set the method to DELETE.
* Set the URI of the object to delete (for example https://localhost:5001/api/todoitems/1).
* Select **Send**.

## Prevent over-posting

Currently the sample app exposes the entire TodoItem object. Production apps typically limit the data that's input and returned using a subset of the model. There are multiple reasons behind this and security is a major one. The subset of a model is usually referred to as a Data Transfer Object (DTO), input model, or view model. **DTO** is used in this article.

A DTO may be used to:

* Prevent over-posting.
* Hide properties that clients are not supposed to view.
* Omit some properties in order to reduce payload size.
* Flatten object graphs that contain nested objects. Flattened object graphs can be more convenient for clients.

To demonstrate the DTO approach, update the TodoItem class to include a secret field:

namespace TodoApi.Models

{

public class TodoItem

{

public long Id { get; set; }

public string Name { get; set; }

public bool IsComplete { get; set; }

public string Secret { get; set; }

}

}

The secret field needs to be hidden from this app, but an administrative app could choose to expose it.

Verify you can post and get the secret field.

Create a DTO model:

public class TodoItemDTO

{

public long Id { get; set; }

public string Name { get; set; }

public bool IsComplete { get; set; }

}

Update the TodoItemsController to use TodoItemDTO:

// GET: api/TodoItems

[HttpGet]

public async Task<ActionResult<IEnumerable<TodoItemDTO>>> GetTodoItems()

{

return await \_context.TodoItems

.Select(x => ItemToDTO(x))

.ToListAsync();

}

[HttpGet("{id}")]

public async Task<ActionResult<TodoItemDTO>> GetTodoItem(long id)

{

var todoItem = await \_context.TodoItems.FindAsync(id);

if (todoItem == null)

{

return NotFound();

}

return ItemToDTO(todoItem);

}

[HttpPut("{id}")]

public async Task<IActionResult> UpdateTodoItem(long id, TodoItemDTO todoItemDTO)

{

if (id != todoItemDTO.Id)

{

return BadRequest();

}

var todoItem = await \_context.TodoItems.FindAsync(id);

if (todoItem == null)

{

return NotFound();

}

todoItem.Name = todoItemDTO.Name;

todoItem.IsComplete = todoItemDTO.IsComplete;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException) when (!TodoItemExists(id))

{

return NotFound();

}

return NoContent();

}

[HttpPost]

public async Task<ActionResult<TodoItemDTO>> CreateTodoItem(TodoItemDTO todoItemDTO)

{

var todoItem = new TodoItem

{

IsComplete = todoItemDTO.IsComplete,

Name = todoItemDTO.Name

};

\_context.TodoItems.Add(todoItem);

await \_context.SaveChangesAsync();

return CreatedAtAction(

nameof(GetTodoItem),

new { id = todoItem.Id },

ItemToDTO(todoItem));

}

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteTodoItem(long id)

{

var todoItem = await \_context.TodoItems.FindAsync(id);

if (todoItem == null)

{

return NotFound();

}

\_context.TodoItems.Remove(todoItem);

await \_context.SaveChangesAsync();

return NoContent();

}

private bool TodoItemExists(long id) =>

\_context.TodoItems.Any(e => e.Id == id);

private static TodoItemDTO ItemToDTO(TodoItem todoItem) =>

new TodoItemDTO

{

Id = todoItem.Id,

Name = todoItem.Name,

IsComplete = todoItem.IsComplete

};

Verify you can't post or get the secret field.