CLDV6212 POE Part 2

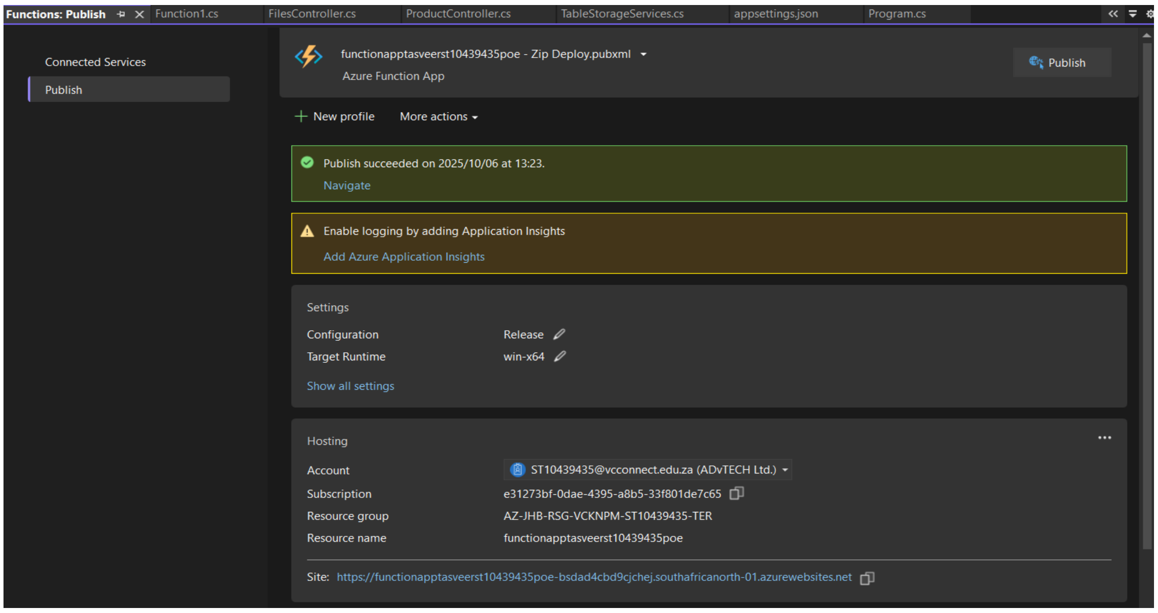
ST10439435

WebApp Link: https://tasveerst10439435part2-degvhzaxfycxfufg.southafricanorth-01.azurewebsites.net

Function App Link: https://functionapptasveerst10439435poe-bsdad4cbd9cjchej.southafricanorth-01.azurewebsites.net

GitHub Link: https://github.com/Veer-14/cloud-development-b-part-1-st10439435.git

YouTube Link: https://youtu.be/rE89FMZkmyo



functionapptasveerst10439435poe - Zip Deploy.pubxml
Azure Function App

Publish succeeded on 2025/10/06 at 13:23.
Navigate

Enable logging by adding Application Insights
Add Azure Application Insights

Settings

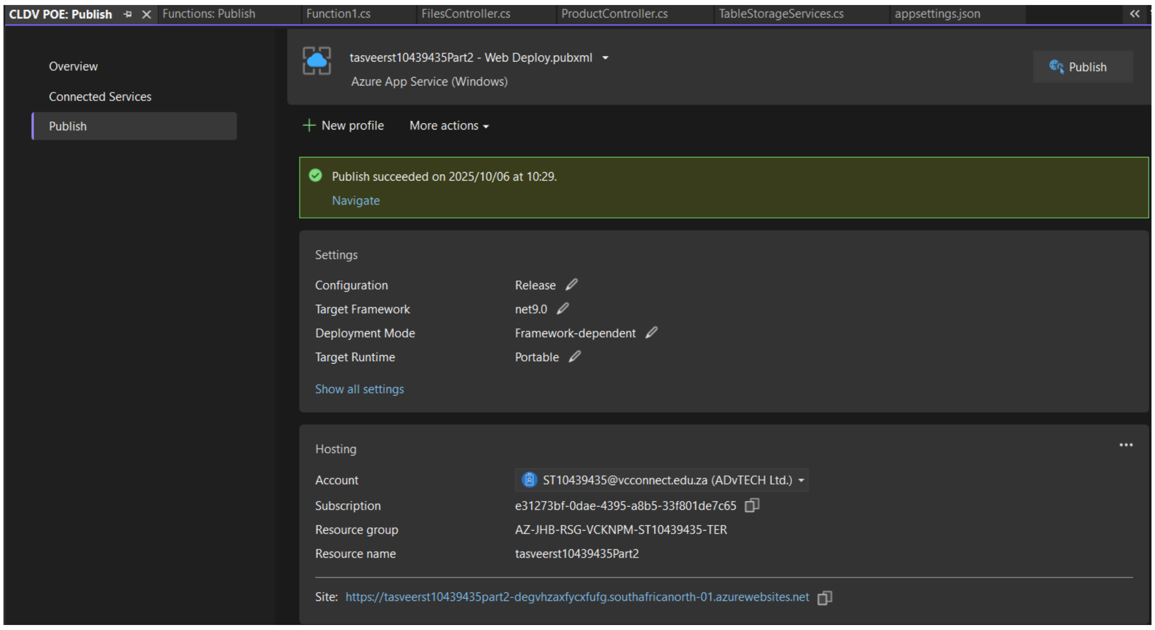
Configuration	Release
Target Runtime	win-x64

Show all settings

Hosting

Account	ST10439435@vcconnect.edu.za (ADvTECH Ltd.)
Subscription	e31273bf-0dae-4395-a8b5-33f801de7c65
Resource group	AZ-JHB-RSG-VCKNPM-ST10439435-TER
Resource name	functionapptasveerst10439435poe

Site: https://functionapptasveerst10439435poe-bsdad4cbd9cjchej.southafricanorth-01.azurewebsites.net



tasveerst10439435Part2 - Web Deploy.pubxml
Azure App Service (Windows)

Publish succeeded on 2025/10/06 at 10:29.
Navigate

Settings

Configuration	Release
Target Framework	net9.0
Deployment Mode	Framework-dependent
Target Runtime	Portable

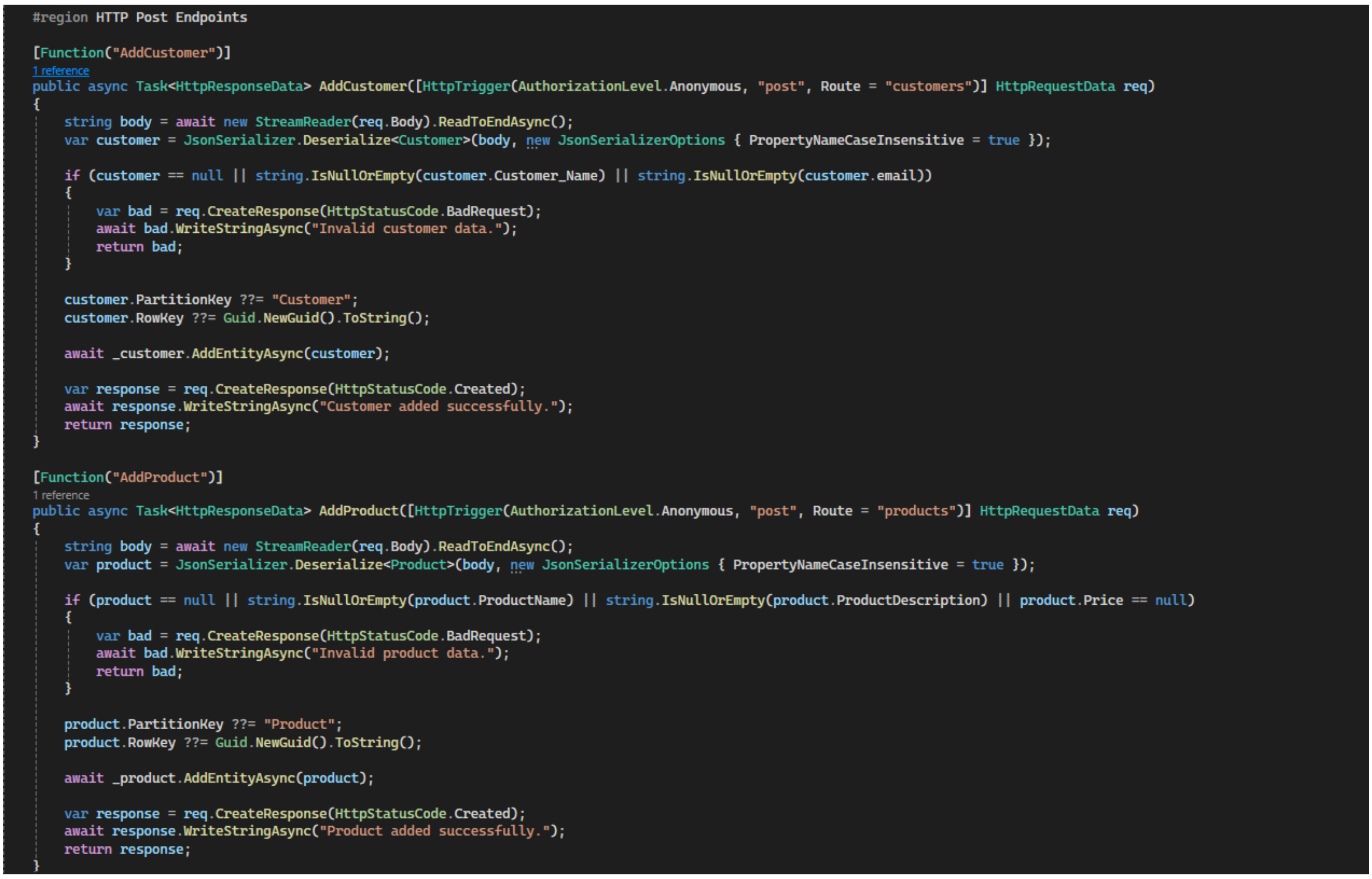
Show all settings

Hosting

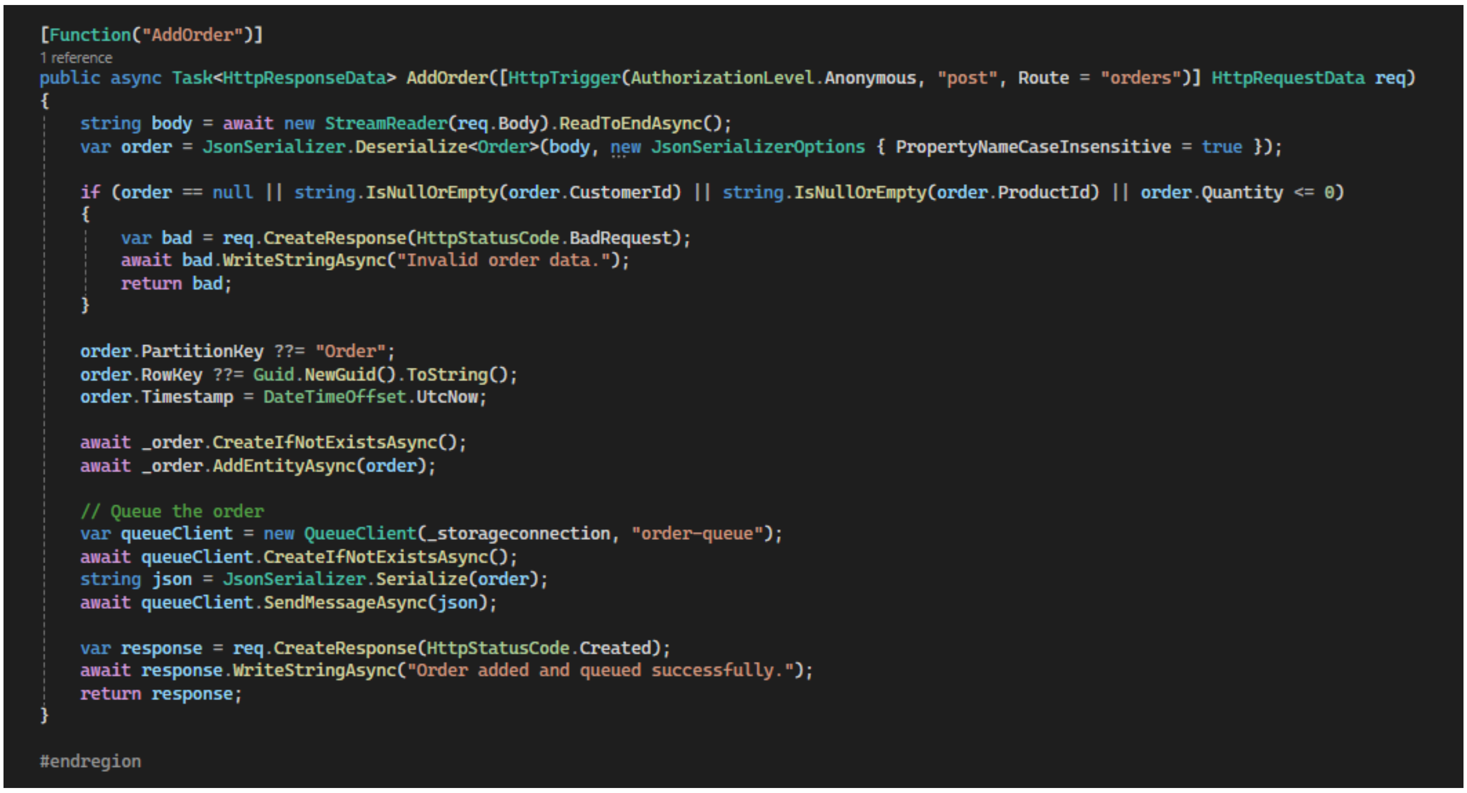
Account	ST10439435@vcconnect.edu.za (ADvTECH Ltd.)
Subscription	e31273bf-0dae-4395-a8b5-33f801de7c65
Resource group	AZ-JHB-RSG-VCKNPM-ST10439435-TER
Resource name	tasveerst10439435Part2

Site: https://tasveerst10439435part2-degvhzaxfycxfufg.southafricanorth-01.azurewebsites.net

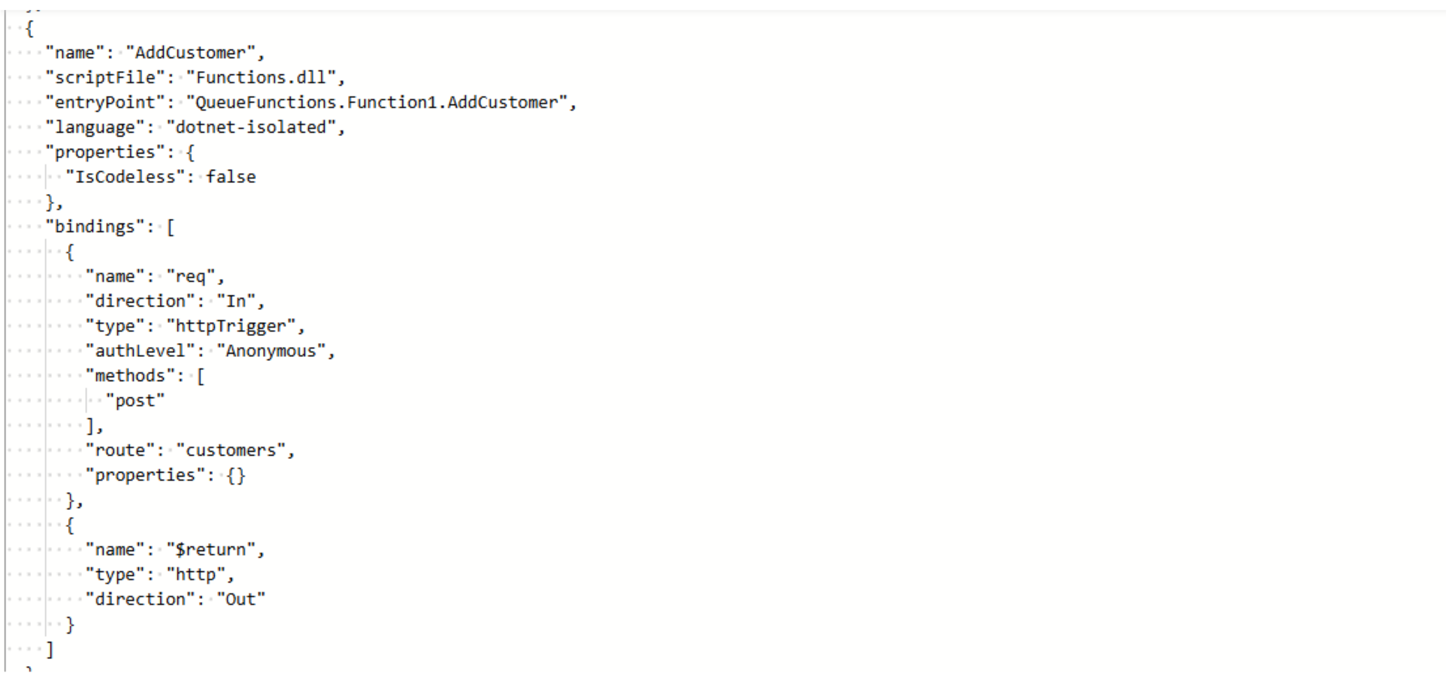
Functions that stores information into azure tables



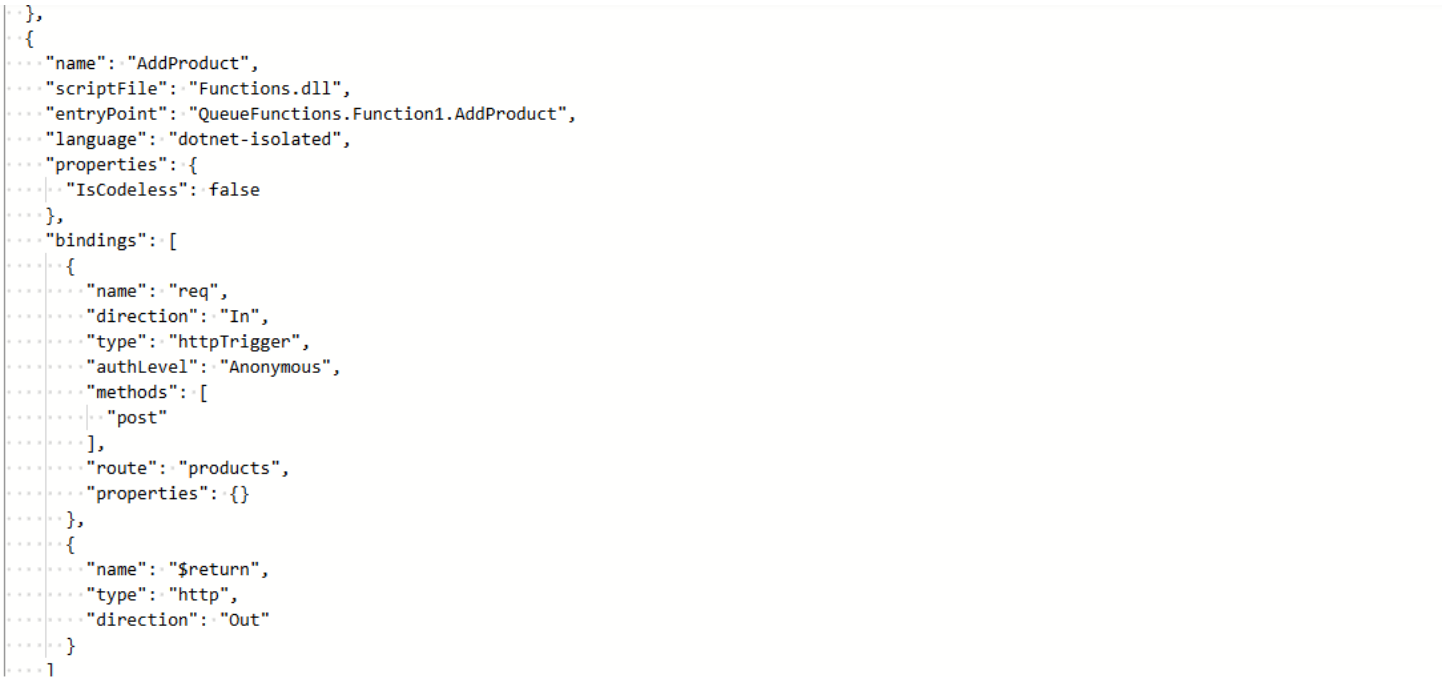
```
#region HTTP Post Endpoints

[Function("AddCustomer")]
1 reference
public async Task<HttpResponseData> AddCustomer([HttpTrigger(AuthorizationLevel.Anonymous, "post", Route = "customers")] HttpRequestData req)
{
    string body = await new StreamReader(req.Body).ReadToEndAsync();
    var customer = JsonSerializer.Deserialize<Customer>(body, new JsonSerializerOptions { PropertyNameCaseInsensitive = true });

    if (customer == null || string.IsNullOrEmpty(customer.Customer_Name) || string.IsNullOrEmpty(customer.email))
    {
        var bad = req.CreateResponse(HttpStatusCode.BadRequest);
        await bad.WriteStringAsync("Invalid customer data.");
        return bad;
    }

    customer.PartitionKey ??= "Customer";
    customer.RowKey ??= Guid.NewGuid().ToString();

    await _customer.AddEntityAsync(customer);

    var response = req.CreateResponse(HttpStatusCode.Created);
    await response.WriteStringAsync("Customer added successfully.");
    return response;
}

[Function("AddProduct")]
1 reference
public async Task<HttpResponseData> AddProduct([HttpTrigger(AuthorizationLevel.Anonymous, "post", Route = "products")] HttpRequestData req)
{
    string body = await new StreamReader(req.Body).ReadToEndAsync();
    var product = JsonSerializer.Deserialize<Product>(body, new JsonSerializerOptions { PropertyNameCaseInsensitive = true });

    if (product == null || string.IsNullOrEmpty(product.ProductName) || string.IsNullOrEmpty(product.ProductDescription) || product.Price == null)
    {
        var bad = req.CreateResponse(HttpStatusCode.BadRequest);
        await bad.WriteStringAsync("Invalid product data.");
        return bad;
    }

    product.PartitionKey ??= "Product";
    product.RowKey ??= Guid.NewGuid().ToString();

    await _product.AddEntityAsync(product);

    var response = req.CreateResponse(HttpStatusCode.Created);
    await response.WriteStringAsync("Product added successfully.");
    return response;
}
```



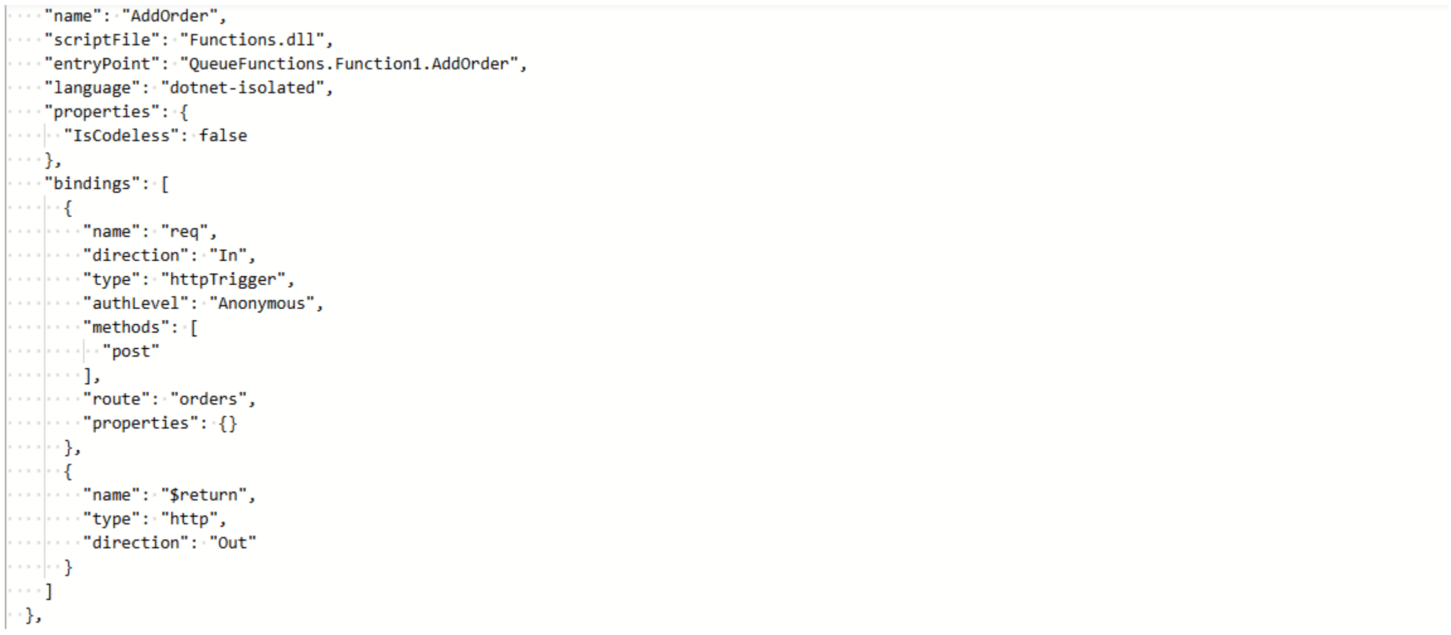
```
[Function("AddOrder")]
1 reference
public async Task<HttpResponseData> AddOrder([HttpTrigger(AuthorizationLevel.Anonymous, "post", Route = "orders")] HttpRequestData req)
{
    string body = await new StreamReader(req.Body).ReadToEndAsync();
    var order = JsonSerializer.Deserialize<Order>(body, new JsonSerializerOptions { PropertyNameCaseInsensitive = true });

    if (order == null || string.IsNullOrEmpty(order.CustomerId) || string.IsNullOrEmpty(order.ProductId) || order.Quantity <= 0)
    {
        var bad = req.CreateResponse(HttpStatusCode.BadRequest);
        await bad.WriteStringAsync("Invalid order data.");
        return bad;
    }

    order.PartitionKey ??= "Order";
    order.RowKey ??= Guid.NewGuid().ToString();
    order.Timestamp = DateTimeOffset.UtcNow;

    await _order.CreateIfNotExistsAsync();
    await _order.AddEntityAsync(order);

    // Queue the order
    var queueClient = new QueueClient(_storageconnection, "order-queue");
    await queueClient.CreateIfNotExistsAsync();
    string json = JsonSerializer.Serialize(order);
    await queueClient.SendMessageAsync(json);

    var response = req.CreateResponse(HttpStatusCode.Created);
    await response.WriteStringAsync("Order added and queued successfully.");
    return response;
}

#endregion
```



```
  {
    "name": "AddCustomer",
    "scriptFile": "Functions.dll",
    "entryPoint": "QueueFunctions.Function1.AddCustomer",
    "language": "dotnet-isolated",
    "properties": {
      "IsCodeless": false
    },
    "bindings": [
      {
        "name": "req",
        "direction": "In",
        "type": "httpTrigger",
        "authLevel": "Anonymous",
        "methods": [
          "post"
        ],
        "route": "customers",
        "properties": {}
      },
      {
        "name": "$return",
        "type": "http",
        "direction": "Out"
      }
    ]
```

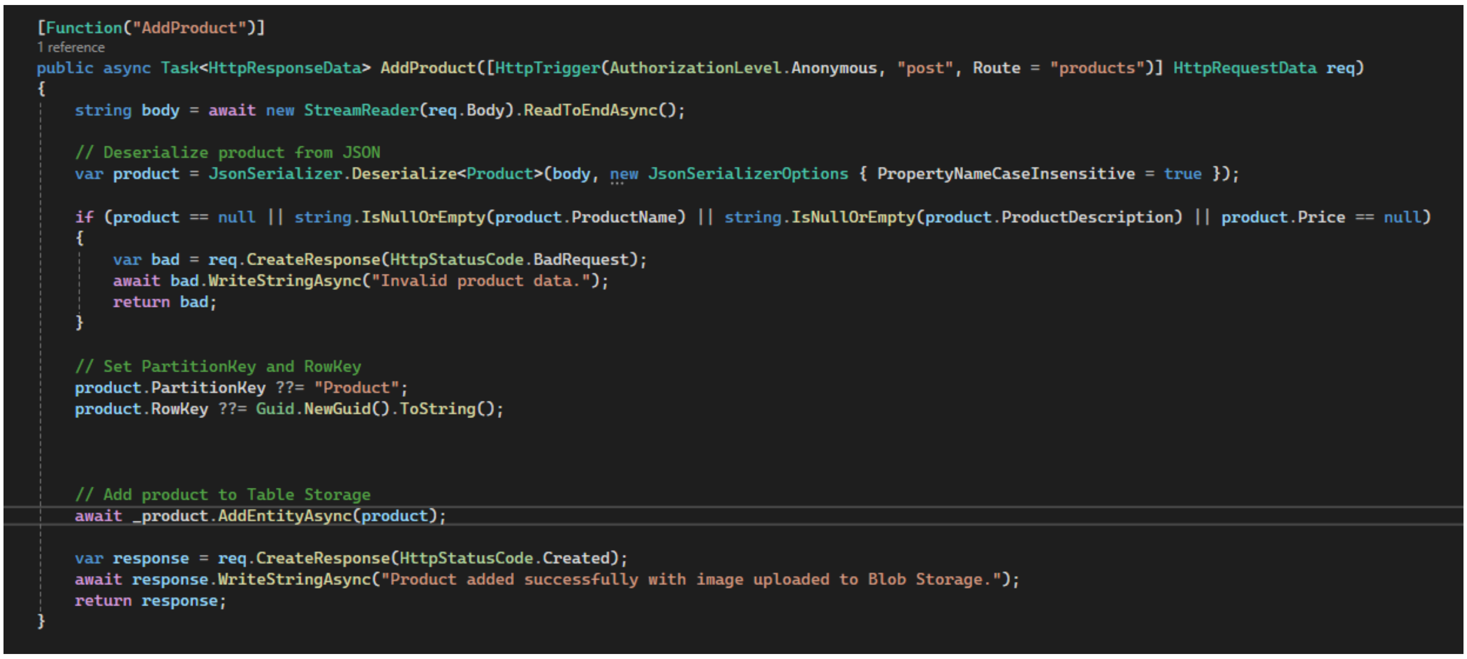


```
  },
  {
    "name": "AddProduct",
    "scriptFile": "Functions.dll",
    "entryPoint": "QueueFunctions.Function1.AddProduct",
    "language": "dotnet-isolated",
    "properties": {
      "IsCodeless": false
    },
    "bindings": [
      {
        "name": "req",
        "direction": "In",
        "type": "httpTrigger",
        "authLevel": "Anonymous",
        "methods": [
          "post"
        ],
        "route": "products",
        "properties": {}
      },
      {
        "name": "$return",
        "type": "http",
        "direction": "Out"
      }
    ]
```

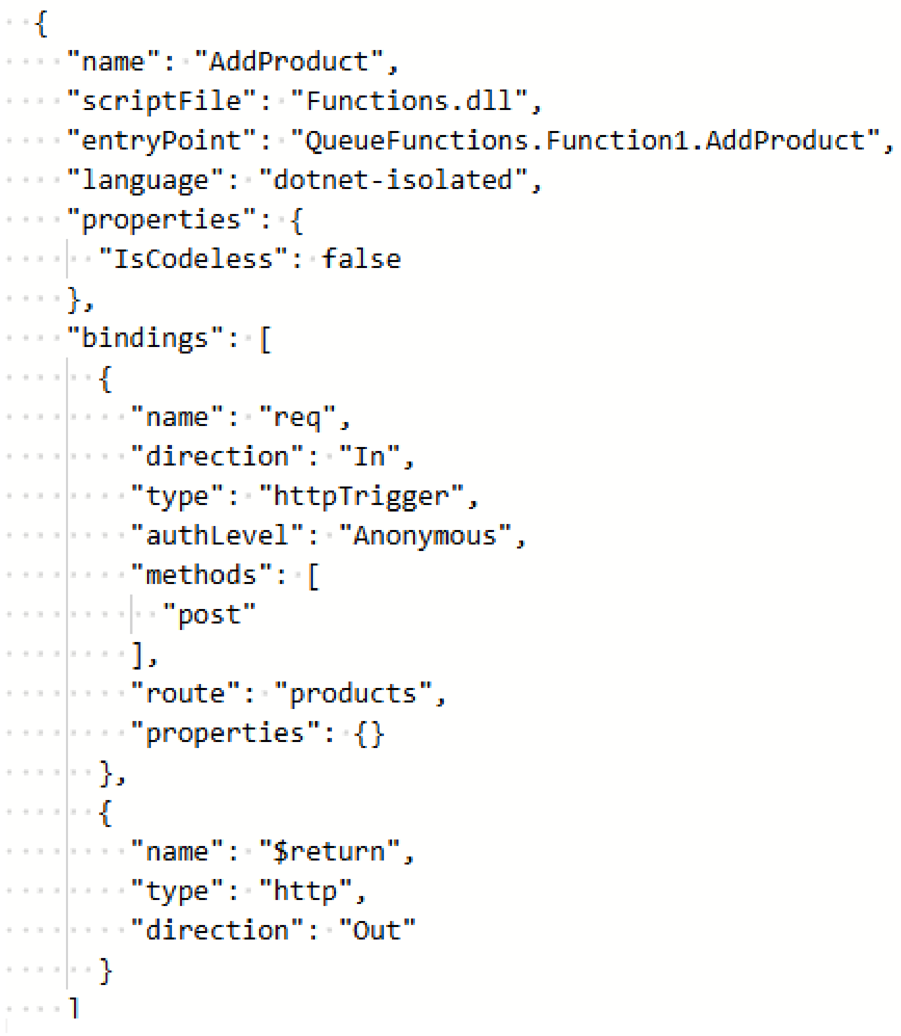


```
    "name": "AddOrder",
    "scriptFile": "Functions.dll",
    "entryPoint": "QueueFunctions.Function1.AddOrder",
    "language": "dotnet-isolated",
    "properties": {
      "IsCodeless": false
    },
    "bindings": [
      {
        "name": "req",
        "direction": "In",
        "type": "httpTrigger",
        "authLevel": "Anonymous",
        "methods": [
          "post"
        ],
        "route": "orders",
        "properties": {}
      },
      {
        "name": "$return",
        "type": "http",
        "direction": "Out"
      }
    ]
  },
```

Function that writes to azure blob storage

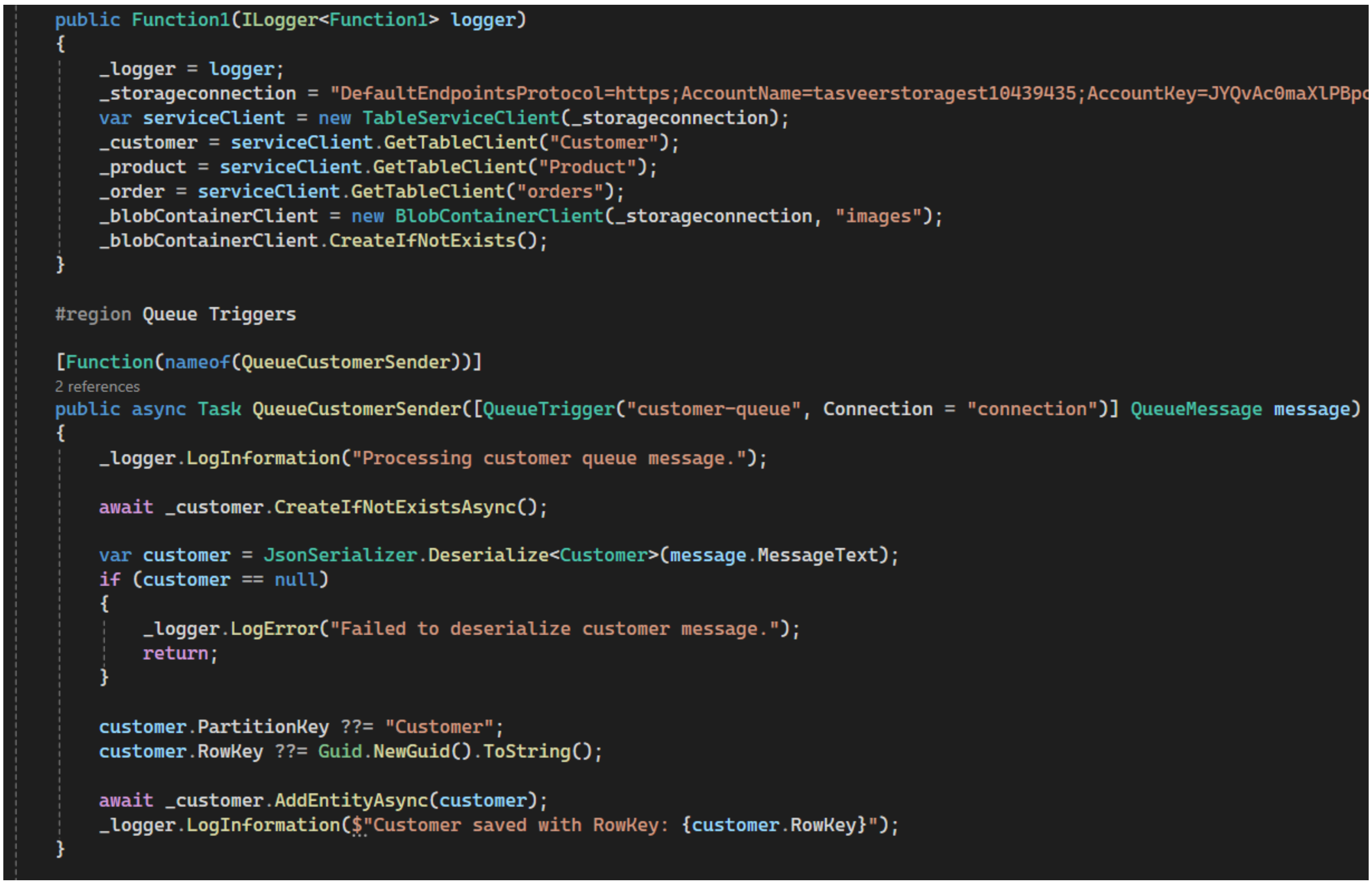


```csharp
[Function("AddProduct")]
1 reference
public async Task<HttpResponseData> AddProduct([HttpTrigger(AuthorizationLevel.Anonymous, "post", Route = "products")] HttpRequestData req)
{
    string body = await new StreamReader(req.Body).ReadToEndAsync();

    // Deserialize product from JSON
    var product = JsonSerializer.Deserialize<Product>(body, new JsonSerializerOptions { PropertyNameCaseInsensitive = true });

    if (product == null || string.IsNullOrEmpty(product.ProductName) || string.IsNullOrEmpty(product.ProductDescription) || product.Price == null)
    {
        var bad = req.CreateResponse(HttpStatusCode.BadRequest);
        await bad.WriteStringAsync("Invalid product data.");
        return bad;
    }

    // Set PartitionKey and RowKey
    product.PartitionKey ??= "Product";
    product.RowKey ??= Guid.NewGuid().ToString();



    // Add product to Table Storage
    await _product.AddEntityAsync(product);

    var response = req.CreateResponse(HttpStatusCode.Created);
    await response.WriteStringAsync("Product added successfully with image uploaded to Blob Storage.");
    return response;
}
```

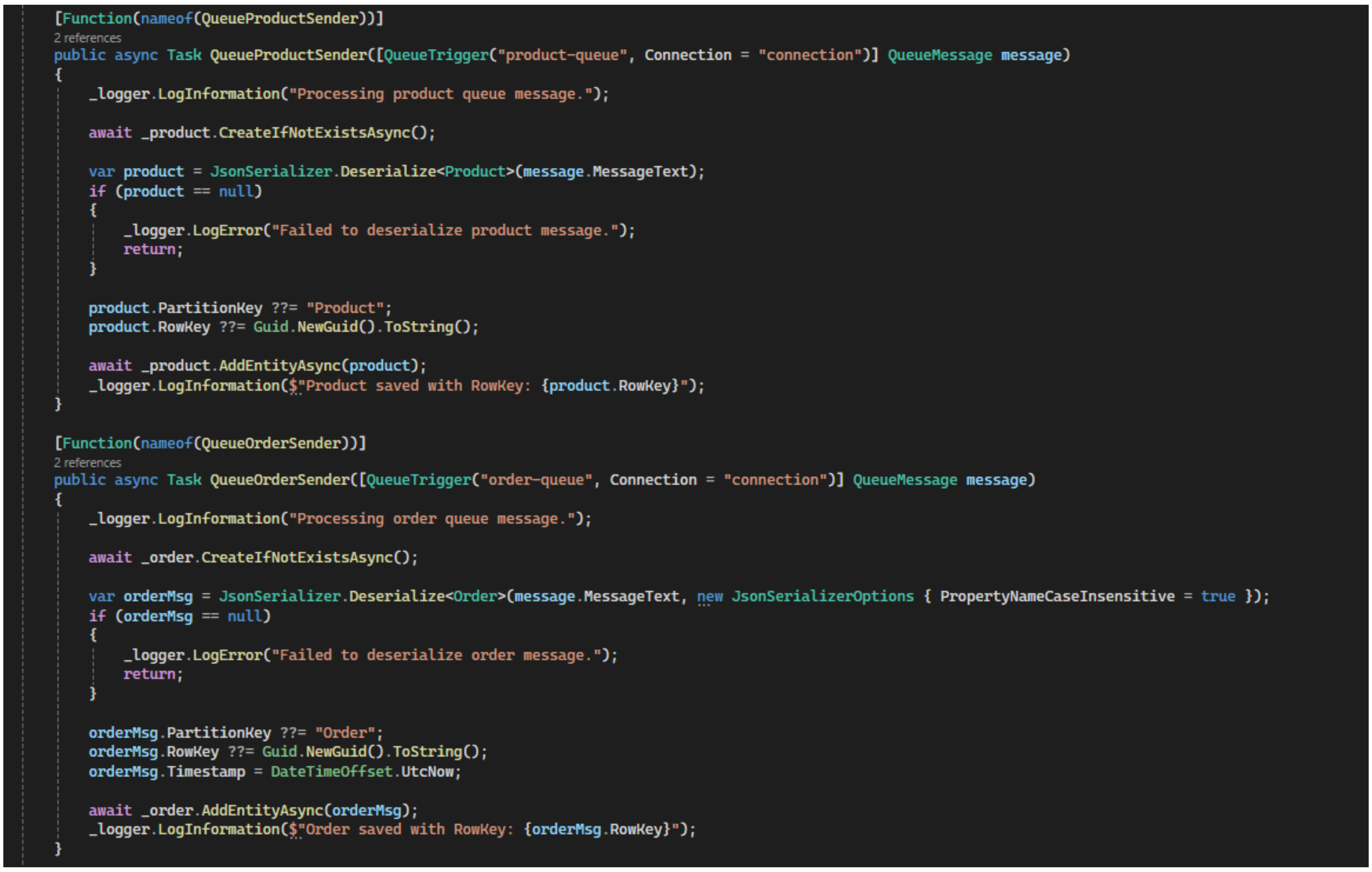


```json
  {
    "name": "AddProduct",
    "scriptFile": "Functions.dll",
    "entryPoint": "QueueFunctions.Function1.AddProduct",
    "language": "dotnet-isolated",
    "properties": {
      "IsCodeless": false
    },
    "bindings": [
      {
        "name": "req",
        "direction": "In",
        "type": "httpTrigger",
        "authLevel": "Anonymous",
        "methods": [
          "post"
        ],
        "route": "products",
        "properties": {}
      },
      {
        "name": "$return",
        "type": "http",
        "direction": "Out"
      }
    ]
```

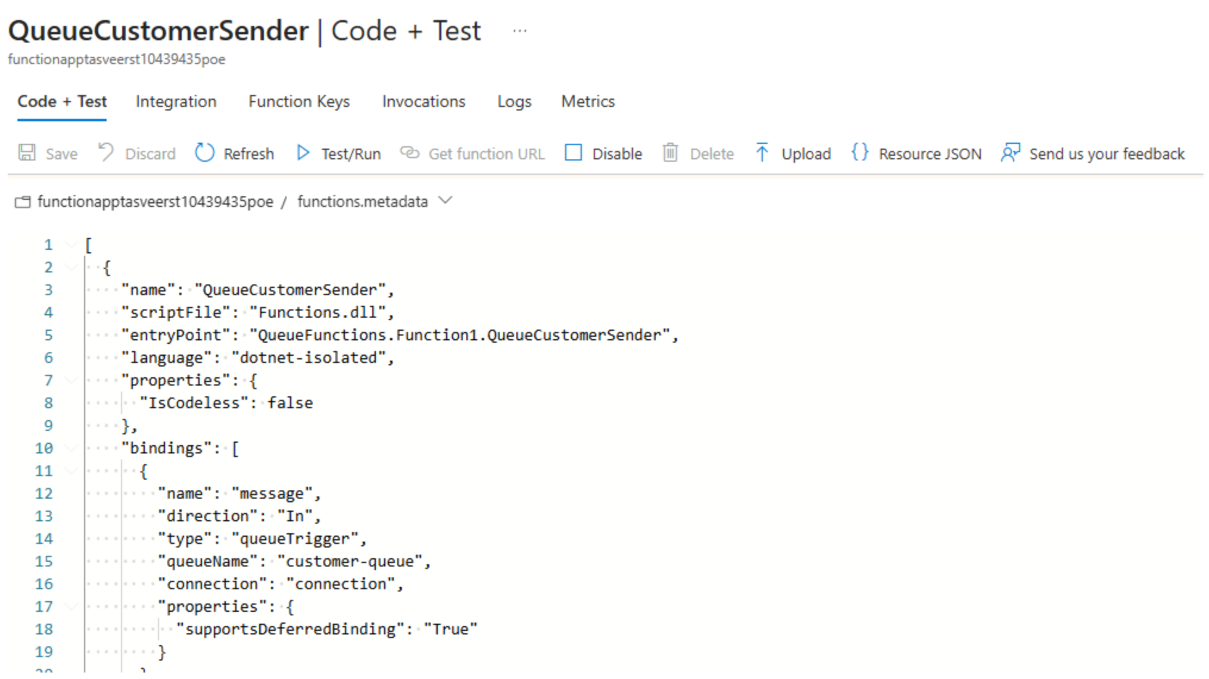
Functions that reads from and writes to azure queues



```csharp
public Function1(ILogger<Function1> logger)
{
    _logger = logger;
    _storageconnection = "DefaultEndpointsProtocol=https;AccountName=tasveerstoragest10439435;AccountKey=JYQvAc0maXlPBp
    var serviceClient = new TableServiceClient(_storageconnection);
    _customer = serviceClient.GetTableClient("Customer");
    _product = serviceClient.GetTableClient("Product");
    _order = serviceClient.GetTableClient("orders");
    _blobContainerClient = new BlobContainerClient(_storageconnection, "images");
    _blobContainerClient.CreateIfNotExists();
}

#region Queue Triggers

[Function(nameof(QueueCustomerSender))]
2 references
public async Task QueueCustomerSender([QueueTrigger("customer-queue", Connection = "connection")] QueueMessage message)
{
    _logger.LogInformation("Processing customer queue message.");

    await _customer.CreateIfNotExistsAsync();

    var customer = JsonSerializer.Deserialize<Customer>(message.MessageText);
    if (customer == null)
    {
        _logger.LogError("Failed to deserialize customer message.");
        return;
    }

    customer.PartitionKey ??= "Customer";
    customer.RowKey ??= Guid.NewGuid().ToString();

    await _customer.AddEntityAsync(customer);
    _logger.LogInformation($"Customer saved with RowKey: {customer.RowKey}");
}
```



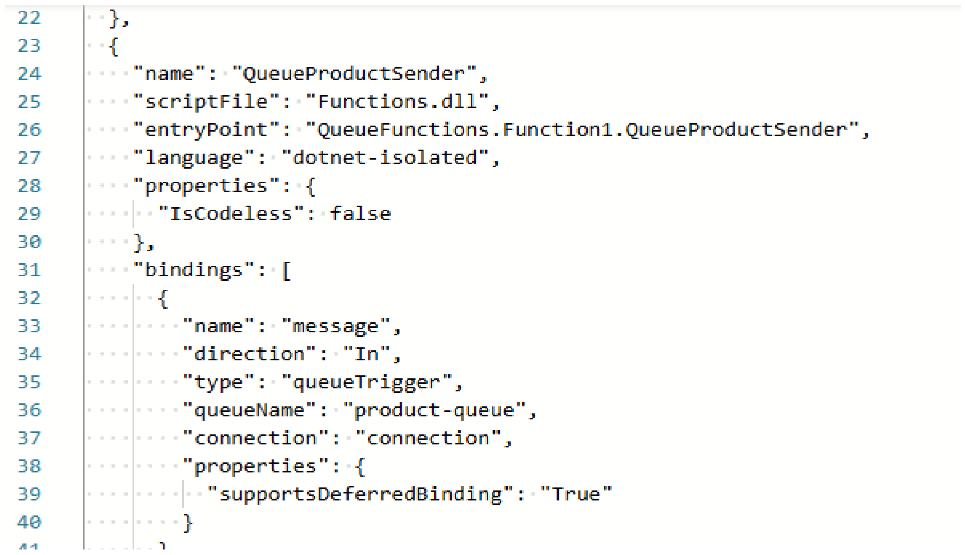
```csharp
[Function(nameof(QueueProductSender))]
2 references
public async Task QueueProductSender([QueueTrigger("product-queue", Connection = "connection")] QueueMessage message)
{
    _logger.LogInformation("Processing product queue message.");

    await _product.CreateIfNotExistsAsync();

    var product = JsonSerializer.Deserialize<Product>(message.MessageText);
    if (product == null)
    {
        _logger.LogError("Failed to deserialize product message.");
        return;
    }

    product.PartitionKey ??= "Product";
    product.RowKey ??= Guid.NewGuid().ToString();

    await _product.AddEntityAsync(product);
    _logger.LogInformation($"Product saved with RowKey: {product.RowKey}");
}

[Function(nameof(QueueOrderSender))]
2 references
public async Task QueueOrderSender([QueueTrigger("order-queue", Connection = "connection")] QueueMessage message)
{
    _logger.LogInformation("Processing order queue message.");

    await _order.CreateIfNotExistsAsync();

    var orderMsg = JsonSerializer.Deserialize<Order>(message.MessageText, new JsonSerializerOptions { PropertyNameCaseInsensitive = true });
    if (orderMsg == null)
    {
        _logger.LogError("Failed to deserialize order message.");
        return;
    }

    orderMsg.PartitionKey ??= "Order";
    orderMsg.RowKey ??= Guid.NewGuid().ToString();
    orderMsg.Timestamp = DateTimeOffset.UtcNow;

    await _order.AddEntityAsync(orderMsg);
    _logger.LogInformation($"Order saved with RowKey: {orderMsg.RowKey}");
}
```


QueueCustomerSender | Code + Test

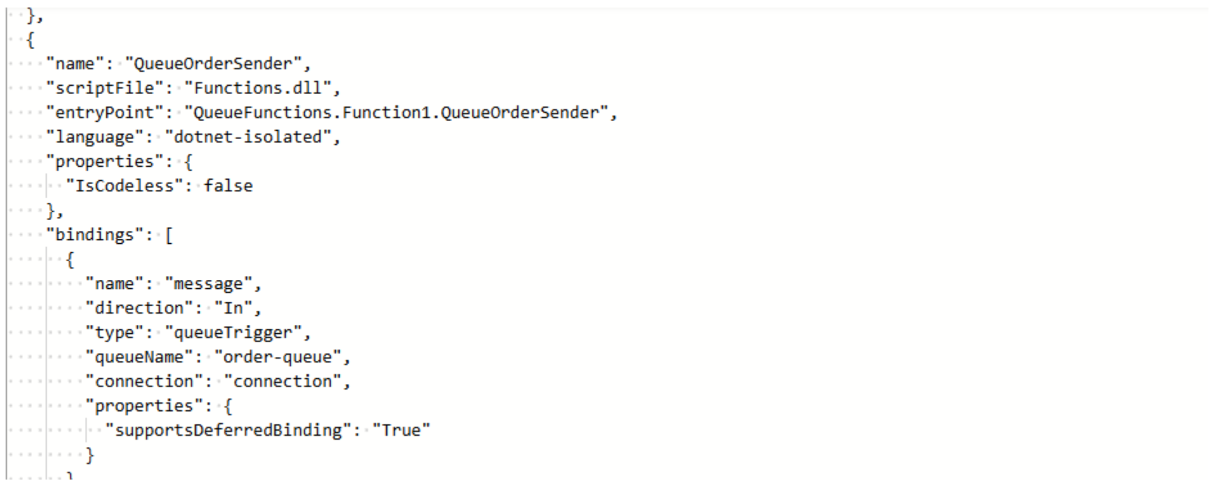
functionapptasveerst10439435poe

functionapptasveerst10439435poe / functions.metadata

```
[
  {
    "name": "QueueCustomerSender",
    "scriptFile": "Functions.dll",
    "entryPoint": "QueueFunctions.Function1.QueueCustomerSender",
    "language": "dotnet-isolated",
    "properties": {
      "IsCodeless": false
    },
    "bindings": [
      {
        "name": "message",
        "direction": "In",
        "type": "queueTrigger",
        "queueName": "customer-queue",
        "connection": "connection",
        "properties": {
          "supportsDeferredBinding": "True"
        }
```

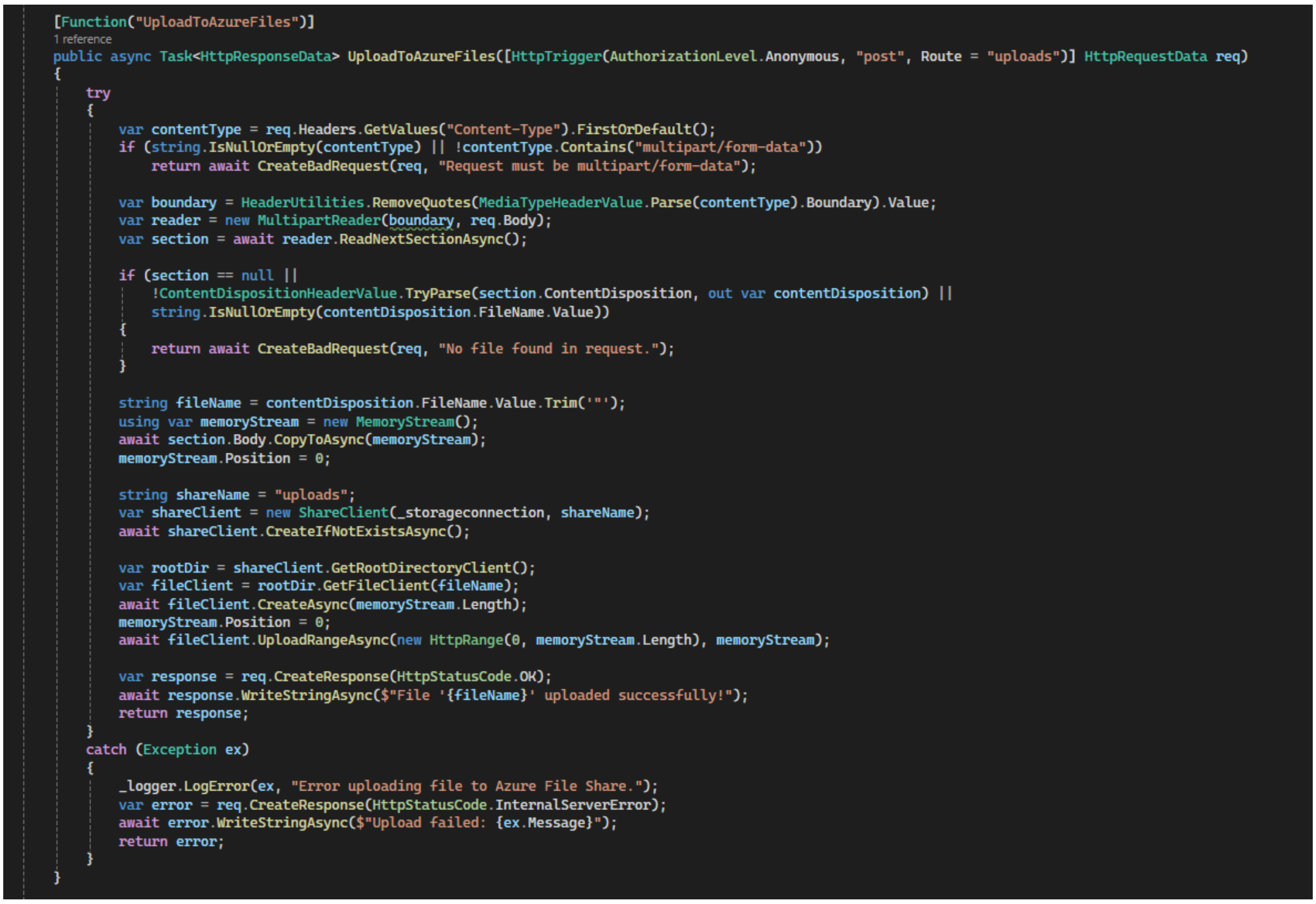


```
  },
  {
    "name": "QueueProductSender",
    "scriptFile": "Functions.dll",
    "entryPoint": "QueueFunctions.Function1.QueueProductSender",
    "language": "dotnet-isolated",
    "properties": {
      "IsCodeless": false
    },
    "bindings": [
      {
        "name": "message",
        "direction": "In",
        "type": "queueTrigger",
        "queueName": "product-queue",
        "connection": "connection",
        "properties": {
          "supportsDeferredBinding": "True"
        }
```

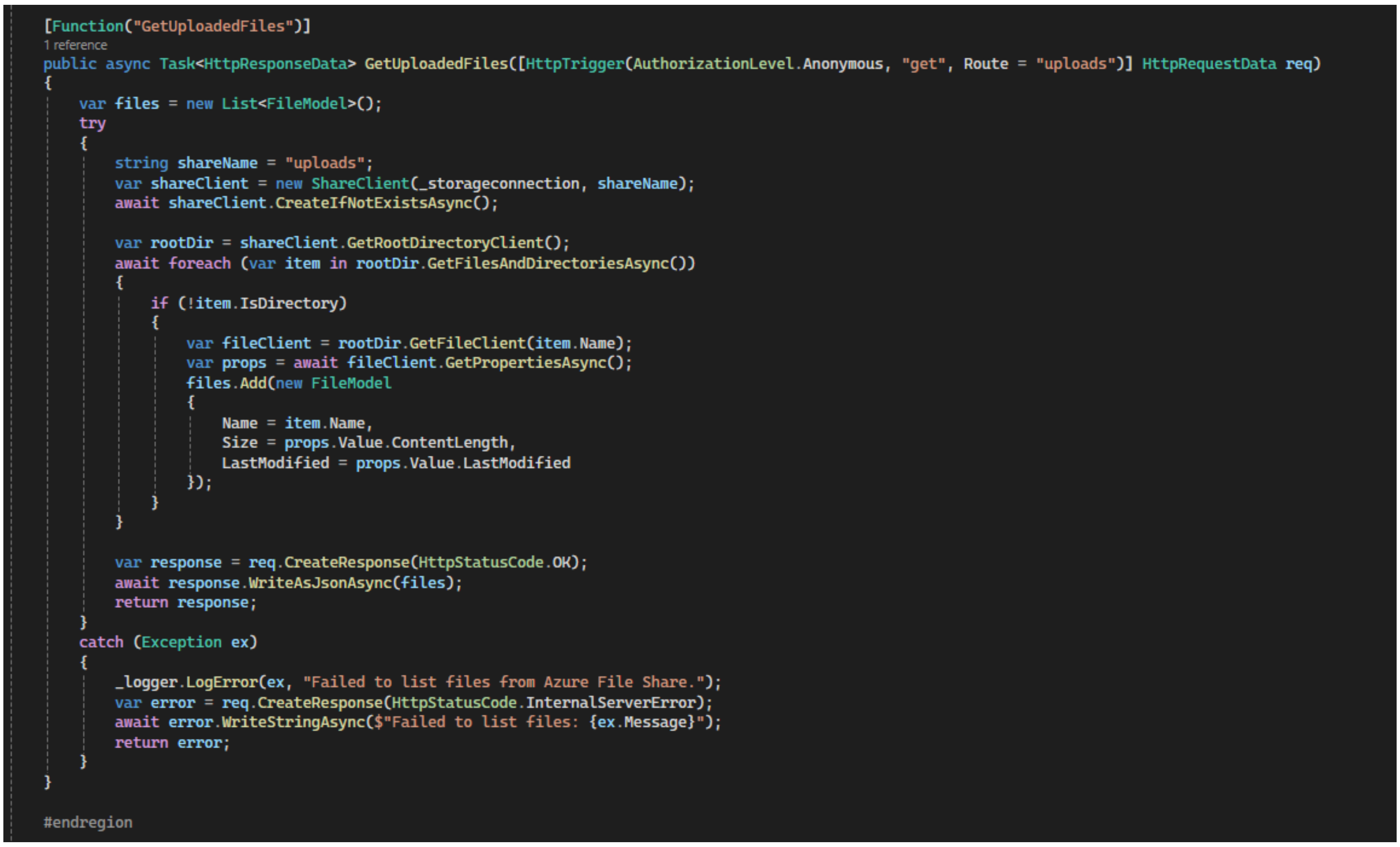


```
  },
  {
    "name": "QueueOrderSender",
    "scriptFile": "Functions.dll",
    "entryPoint": "QueueFunctions.Function1.QueueOrderSender",
    "language": "dotnet-isolated",
    "properties": {
      "IsCodeless": false
    },
    "bindings": [
      {
        "name": "message",
        "direction": "In",
        "type": "queueTrigger",
        "queueName": "order-queue",
        "connection": "connection",
        "properties": {
          "supportsDeferredBinding": "True"
        }
```

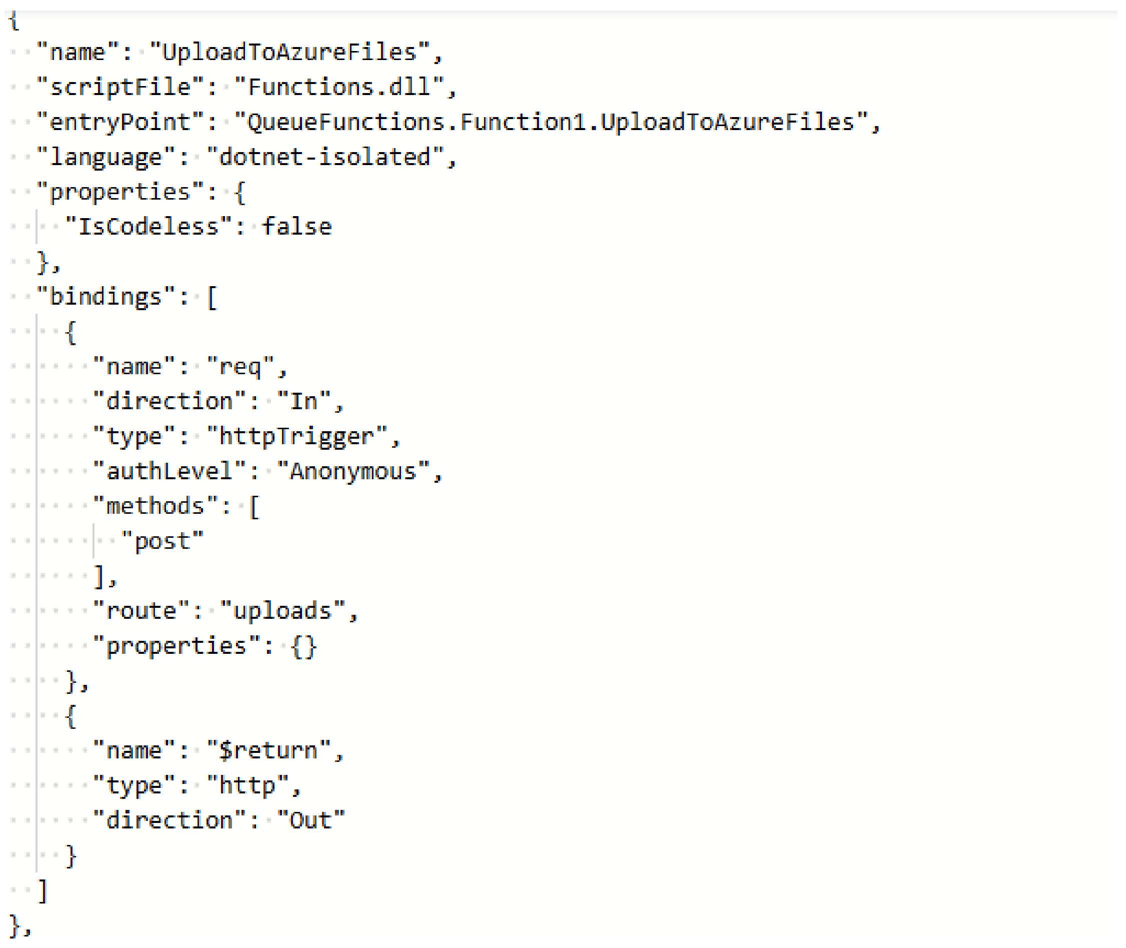
Functions that sends a file to azure files



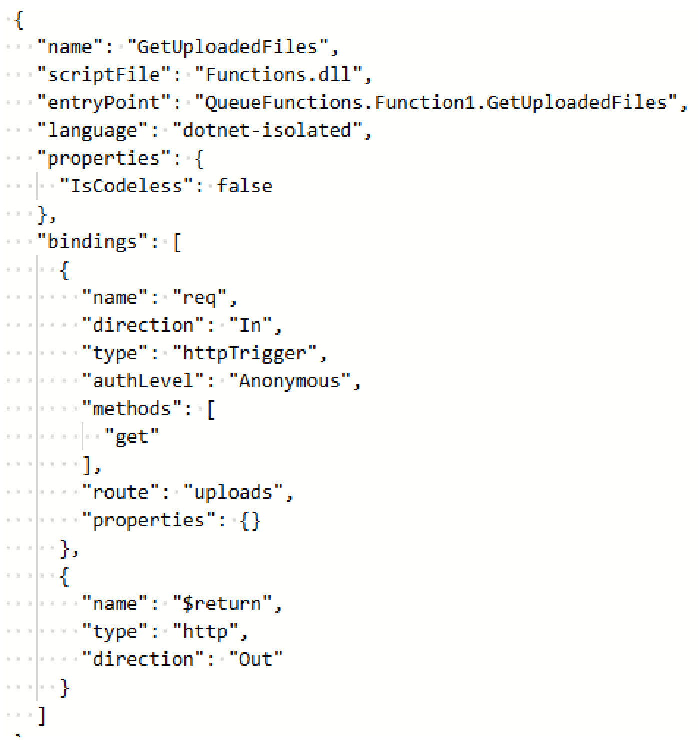
```
[Function("UploadToAzureFiles")]
1 reference
public async Task<HttpResponseData> UploadToAzureFiles([HttpTrigger(AuthorizationLevel.Anonymous, "post", Route = "uploads")] HttpRequestData req)
{
    try
    {
        var contentType = req.Headers.GetValues("Content-Type").FirstOrDefault();
        if (string.IsNullOrEmpty(contentType) || !contentType.Contains("multipart/form-data"))
            return await CreateBadRequest(req, "Request must be multipart/form-data");

        var boundary = HeaderUtilities.RemoveQuotes(MediaTypeHeaderValue.Parse(contentType).Boundary).Value;
        var reader = new MultipartReader(boundary, req.Body);
        var section = await reader.ReadNextSectionAsync();

        if (section == null ||
            !ContentDispositionHeaderValue.TryParse(section.ContentDisposition, out var contentDisposition) ||
            string.IsNullOrEmpty(contentDisposition.FileName.Value))
        {
            return await CreateBadRequest(req, "No file found in request.");
        }

        string fileName = contentDisposition.FileName.Value.Trim('"');
        using var memoryStream = new MemoryStream();
        await section.Body.CopyToAsync(memoryStream);
        memoryStream.Position = 0;

        string shareName = "uploads";
        var shareClient = new ShareClient(_storageconnection, shareName);
        await shareClient.CreateIfNotExistsAsync();

        var rootDir = shareClient.GetRootDirectoryClient();
        var fileClient = rootDir.GetFileClient(fileName);
        await fileClient.CreateAsync(memoryStream.Length);
        memoryStream.Position = 0;
        await fileClient.UploadRangeAsync(new HttpRange(0, memoryStream.Length), memoryStream);

        var response = req.CreateResponse(HttpStatusCode.OK);
        await response.WriteStringAsync($"File '{fileName}' uploaded successfully!");
        return response;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error uploading file to Azure File Share.");
        var error = req.CreateResponse(HttpStatusCode.InternalServerError);
        await error.WriteStringAsync($"Upload failed: {ex.Message}");
        return error;
    }
}
```



```
[Function("GetUploadedFiles")]
1 reference
public async Task<HttpResponseData> GetUploadedFiles([HttpTrigger(AuthorizationLevel.Anonymous, "get", Route = "uploads")] HttpRequestData req)
{
    var files = new List<FileModel>();
    try
    {
        string shareName = "uploads";
        var shareClient = new ShareClient(_storageconnection, shareName);
        await shareClient.CreateIfNotExistsAsync();

        var rootDir = shareClient.GetRootDirectoryClient();
        await foreach (var item in rootDir.GetFilesAndDirectoriesAsync())
        {
            if (!item.IsDirectory)
            {
                var fileClient = rootDir.GetFileClient(item.Name);
                var props = await fileClient.GetPropertiesAsync();
                files.Add(new FileModel
                {
                    Name = item.Name,
                    Size = props.Value.ContentLength,
                    LastModified = props.Value.LastModified
                });
            }
        }

        var response = req.CreateResponse(HttpStatusCode.OK);
        await response.WriteAsJsonAsync(files);
        return response;
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Failed to list files from Azure File Share.");
        var error = req.CreateResponse(HttpStatusCode.InternalServerError);
        await error.WriteStringAsync($"Failed to list files: {ex.Message}");
        return error;
    }
}

#endregion
```



```json
{
  "name": "UploadToAzureFiles",
  "scriptFile": "Functions.dll",
  "entryPoint": "QueueFunctions.Function1.UploadToAzureFiles",
  "language": "dotnet-isolated",
  "properties": {
    "IsCodeless": false
  },
  "bindings": [
    {
      "name": "req",
      "direction": "In",
      "type": "httpTrigger",
      "authLevel": "Anonymous",
      "methods": [
        "post"
      ],
      "route": "uploads",
      "properties": {}
    },
    {
      "name": "$return",
      "type": "http",
      "direction": "Out"
    }
  ]
},
```



```json
{
  "name": "GetUploadedFiles",
  "scriptFile": "Functions.dll",
  "entryPoint": "QueueFunctions.Function1.GetUploadedFiles",
  "language": "dotnet-isolated",
  "properties": {
    "IsCodeless": false
  },
  "bindings": [
    {
      "name": "req",
      "direction": "In",
      "type": "httpTrigger",
      "authLevel": "Anonymous",
      "methods": [
        "get"
      ],
      "route": "uploads",
      "properties": {}
    },
    {
      "name": "$return",
      "type": "http",
      "direction": "Out"
    }
  ]
```

Azure Event Hubs

1. Service Overview

Azure Event Hubs is a streaming service of big data and event ingestion provided by Microsoft Azure. It can ingest and process millions of events in real-time from sensors, devices, and applications. It is designed to process high-throughput data streams that may be processed or analyzed by downstream systems such as Azure Stream Analytics, Databricks, or Power BI. (Kansara)

As Softweb Solutions (n.d.) explains, Event Hubs is appropriate for telemetry, real-time dashboards, and analytics because it supports continuous data stream from multiple sources. (Kansara)

2. Mechanism

Event Hubs is built on a publish–subscribe mechanism where event producers send data into an Event Hub, and multiple consumers (subscribers) read and process the data. (Kansara)

Producers: These are applications, devices, or APIs that push event data into the Event Hub.

- Event Hub (broker in between): Temporarily buffers the incoming events in partitions for scalability as well as parallel consumption.
- Consumers: Applications, analysis engines, or services that consume and process the data streams in real-time.

The service offers partitioning to allow processing of huge loads, consumer groups for multiple requirements of processing, and checkpointing in a way that consumers can resume processing from where they left. (Kansara)

This real-time event stream allows for immediate processing and insights without delay from batch uploads. (Kansara)

3. How it Adds Value to End Users

Event Hubs adds customer happiness in the form of real-time feedback and personalization in your app. (Kansara)

- Real-time feedback: Based on whatever the user is doing when they are logged into the app, their data can be streamed and processed in real time in an effort to personalize content or recommendations.
- Increased reliability: Event Hubs can grow quickly to manage abrupt spikes in usage, ensuring the app is responsive and quick.

- Problem detection: Real-time telemetry allows developers to spot potential application crashes or an performance issues as they happen, allowing for quicker fixes and enhanced user interfaces.
- Engagement: By analyzing patterns in behavior as they occur, the application customizes its interface and suggestions for individual users, which increases engagement and user satisfaction.
  (Kansara)

Overall, Event Hubs provides a data-driven, scalable infrastructure for dynamic, responsive, and consistent user experiences. (Kansara)

Azure Service Bus

1. Service Description

Azure Service Bus is a fully managed enterprise message broker that supports trustable messaging between different parts of an application. It ensures single-shot and in-order delivery of messages even if certain systems are temporarily unavailable. (Kansara)

According to Softweb Solutions (n.d.), Service Bus is best used for transactional, ordered, and mission-critical message delivery and is thus best applied to business processes, order processing, and system integration. (Kansara)

2. Mechanism

Service Bus uses a message queue and topic/subscription pattern:

- Queues: Work on a one-to-one basis. A sender sends messages into the queue, and a receiver dequeues and processes them one by one.
- Topics and Subscriptions: Work on a one-to-many basis. A message published on a topic is delivered to many subscriptions (recipients), making it possible for publish–subscribe communication.
- Advanced capabilities: It also supports ordered message delivery sessions, dead-letter queues for handling messages that could not be processed, scheduled delivery, and transactional processing.
  (Kansara)

The mechanism provides for safe app component communication — messages never get lost, even if part of the system fails temporarily. (Kansara)

3. How it Enhances Value to End Users

Azure Service Bus improves customer experience by making every app transaction and notification reliable, ordered, and predictable, even at peak load. (Kansara)

- Reliability: Ensures critical communication (e.g., payment, orders, reservations) is processed once and in order, without duplicates or lost transactions.
- Smooth workflows: Customers receive quicker and more determinable results because backend services communicate smoothly through queues.
- Error handling: Graceful handling of errors through retries and dead-letter queues prevents downtime or perceptible app failures.
- Scalability: Provides your app to scale high traffic without lagging and ensuring responses remain quick to users.
- Notifications: Service Bus topics can give real-time updates or acknowledgments to users, creating trust and engagement.
  (Kansara)

In short, Service Bus enhances reliability, stability, and responsiveness that improve the customer experience directly. (Kansara)

References

Kansara, Romil. "Azure Event Hub vs. Azure Service Bus: A Comprehensive Comparison." *Softwebsolutions*, softwebsolutions, 17 Aug. 2023, www.softwebsolutions.com/resources/azure-service-bus-vs-event-hub.html.