API Testing with RestSharp and Postman

Overview

This guide covers essential concepts and techniques for automating API tests using RestSharp in C# and Postman. API testing is a critical component of modern software testing, allowing you to validate the functionality, reliability, security, and performance of your application's backend services.

Table of Contents

Introduction to API Testing

What is API Testing?

API (Application Programming Interface) testing involves testing APIs directly and as part of integration testing to verify they meet functionality, reliability, performance, and security requirements. Unlike UI testing, API testing focuses on the business logic layer of software architecture.

Benefits of API Testing

- **Early Testing**: Test application logic before the UI is ready
- **Efficiency**: Faster and less resource-intensive than UI tests
- **Stability**: Not affected by UI changes
- **Integration Verification**: Ensures different components work together
- **Language-Independent**: Test services built in any language

Common API Testing Types

- **Functionality testing**: Verifying specific functions of the API
- **Reliability testing**: Testing API behavior under various conditions
- **Load testing**: Checking performance under expected load
- **Security testing**: Identifying vulnerabilities
- **Usability testing**: Ensuring the API is easy to use

Setting Up RestSharp

RestSharp is a popular HTTP client library for .NET, making it easy to interact with and test RESTful APIs.

Installation

1. Create a new C# test project (using NUnit, MSTest, or xUnit)
2. Install RestSharp via NuGet Package Manager:

```
// Using Package Manager Console
Install-Package RestSharp

// Using .NET CLI
dotnet add package RestSharp
```

3. Install JSON serialization support:

```
Install-Package Newtonsoft.Json
// OR
Install-Package System.Text.Json
```

Basic Setup Example

```
using NUnit.Framework;
using RestSharp;
using System.Net;
using Newtonsoft.Json;

namespace ApiTests
{
    [TestFixture]
    public class ApiTests
    {
        private RestClient _client;
        private const string BaseUrl = "https://fakestoreapi.com";

        [SetUp]
        public void Setup()
        {
            // Initialize RestClient
            _client = new RestClient(BaseUrl);
        }

        [Test]
        public void SampleApiTest()
        {
            // Create request
            var request = new RestRequest("/products", Method.Get);

            // Execute request
            var response = _client.Execute(request);

            // Assert
            Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK));
```

```
        }
    }
}
```

RestClient and RestRequest in Detail

RestSharp's core functionality is centered around two main classes: `RestClient` and `RestRequest`. Understanding these classes and their members is essential for effective API testing.

RestClient Class

The `RestClient` class is the primary class for making HTTP requests in RestSharp. It handles the connection, sending requests, and processing responses.

Important Properties and Methods

Member	Description
`BaseUrl`	Gets or sets the base URL for requests
`DefaultParameters`	Collection of parameters sent with every request
`Authenticator`	Gets or sets the authenticator used for requests
`UserAgent`	Gets or sets the user agent for requests
`Timeout`	Gets or sets the timeout in milliseconds
`Execute(IRestRequest)`	Executes a request and returns a response
`ExecuteAsync(IRestRequest)`	Asynchronously executes a request
`Get<T>(IRestRequest)`	Executes a GET request and deserializes to specified type
`Post<T>(IRestRequest)`	Executes a POST request and deserializes to specified type
`Put<T>(IRestRequest)`	Executes a PUT request and deserializes to specified type
`Delete<T>(IRestRequest)`	Executes a DELETE request and deserializes to specified type

RestClient Configuration Examples

```
// Basic configuration
var client = new RestClient("https://api.example.com");
client.Timeout = 10000; // 10 seconds

// Using default parameters (sent with every request)
client.DefaultParameters.Add(new Parameter("apiKey", "your-api-key",
ParameterType.QueryString));

// Setting up authentication
client.Authenticator = new HttpBasicAuthenticator("username", "password");
```

```
// OR
client.Authenticator = new JwtAuthenticator("your-jwt-token");

// Configuring SSL/TLS
client.ConfigureWebRequest(request => {
    request.ServicePoint.SecurityProtocol = SecurityProtocolType.Tls12;
});
```

RestRequest Class

The `RestRequest` class represents an HTTP request that can be executed by a `RestClient`. It allows you to specify the resource, method, parameters, headers, and body.

Important Properties and Methods

Member	Description
`Resource`	Gets or sets the resource (URI) for the request
`Method`	Gets or sets the HTTP method (GET, POST, etc.)
`Parameters`	Collection of parameters for the request
`AddParameter()`	Adds a parameter to the request
`AddHeader()`	Adds a header to the request
`AddHeaders()`	Adds multiple headers to the request
`AddQueryParameter()`	Adds a query parameter to the request URL
`AddUrlSegment()`	Adds a URL segment parameter
`AddBody()`	Adds an object as the body, serialized as JSON or XML
`AddJsonBody()`	Adds an object as the request body, serialized to JSON
`AddFile()`	Adds a file to the request
`AddCookie()`	Adds a cookie to the request
`OnBeforeDeserialization`	Event that fires before response deserialization

RestRequest Configuration Examples

```
// Basic request
var request = new RestRequest("api/users", Method.GET);

// URL segments (for dynamic URLs)
request = new RestRequest("api/users/{id}", Method.GET);
request.AddUrlSegment("id", 123);
// Result: api/users/123
```

```
// Query parameters
request.AddQueryParameter("page", "1");
request.AddQueryParameter("sort", "name");
// Result: api/users?page=1&sort=name

// Headers
request.AddHeader("Accept", "application/json");
request.AddHeader("Authorization", "Bearer token123");

// Multiple headers
var headers = new Dictionary<string, string>
{
    { "X-Custom-Header", "value" },
    { "X-Another-Header", "another-value" }
};
request.AddHeaders(headers);

// Request body (for POST, PUT)
var user = new User { Name = "John", Email = "john@example.com" };

// JSON body (application/json)
request.AddJsonBody(user);

// Form data (application/x-www-form-urlencoded)
request.AddParameter("name", "John");
request.AddParameter("email", "john@example.com");

// Files
request.AddFile("avatar", "/path/to/image.jpg", "image/jpeg");
```

Response Handling

The `IRestResponse` interface (and its implementation `RestResponse`) represents an HTTP response from a REST request.

Important Properties

Property	Description
`Content`	Gets the content of the response
`ContentType`	Gets the content type of the response
`ContentLength`	Gets the content length of the response
`StatusCode`	Gets the HTTP status code of the response
`IsSuccessful`	Gets a value indicating whether the request was successful
`ErrorMessage`	Gets the error message of the response
`ErrorException`	Gets the exception thrown during the request, if any

Property	Description
`Headers`	Gets the headers of the response
`Cookies`	Gets the cookies of the response
`ResponseUri`	Gets the URI of the response
`ResponseStatus`	Gets the status of the response

Response Handling Examples

```
// Basic response handling
var response = client.Execute(request);
if (response.IsSuccessful)
{
    // Process successful response
    var content = response.Content;
}
else
{
    // Handle error
    Console.WriteLine($"Error: {response.ErrorMessage}");
}

// Deserializing response to a specific type
var response = client.Execute<User>(request);
if (response.IsSuccessful)
{
    var user = response.Data;
    Console.WriteLine($"User name: {user.Name}");
}

// Async response handling with generic type
var response = await client.ExecuteAsync<List<User>>(request);
if (response.IsSuccessful)
{
    foreach (var user in response.Data)
    {
        Console.WriteLine($"User: {user.Name}");
    }
}
```

Data Serialization and Deserialization

RestSharp handles serialization and deserialization of request and response data. By default, it uses its own JSON serializer, but you can also configure it to use other serializers like Newtonsoft.Json.

```
// Using Newtonsoft.Json serializer
var client = new RestClient("https://api.example.com");
```

```
client.UseNewtonsoftJson(); // Extension method from
RestSharp.Serializers.NewtonsoftJson package

// Deserializing manually
var response = client.Execute(request);
if (response.IsSuccessful)
{
    var user = JsonConvert.DeserializeObject<User>(response.Content);
}
```

Writing Tests for CRUD Operations

GET Requests (Read)

```
[Test]
public void GetResourceById_ReturnsCorrectResource()
{
    // Create request
    var request = new RestRequest($"api/users/1", Method.Get);

    // Execute request
    var response = _client.Execute(request);

    // Assert
    Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK));

    // Deserialize response
    var user = JsonConvert.DeserializeObject<User>(response.Content);
    Assert.That(user.Id, Is.EqualTo(1));
}
```

POST Requests (Create)

```
[Test]
public void CreateNewUser_ReturnsCreatedStatus()
{
    // Create new user object
    var newUser = new User
    {
        Name = "John Doe",
        Email = "john.doe@example.com"
    };

    // Create request
    var request = new RestRequest("api/users", Method.Post);
    request.AddJsonBody(newUser);

    // Execute request
    var response = _client.Execute(request);
```

```
    // Assert
    Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.Created));

    // Verify the created user
    var createdUser = JsonConvert.DeserializeObject<User>(response.Content);
    Assert.That(createdUser.Name, Is.EqualTo("John Doe"));
    Assert.That(createdUser.Id, Is.Not.EqualTo(0));
}
```

PUT Requests (Update)

```
[Test]
public void UpdateUser_ReturnsSuccess()
{
    // Updated user data
    var updatedUser = new User
    {
        Id = 1,
        Name = "John Updated",
        Email = "john.updated@example.com"
    };

    // Create request
    var request = new RestRequest($"api/users/1", Method.Put);
    request.AddJsonBody(updatedUser);

    // Execute request
    var response = _client.Execute(request);

    // Assert
    Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK));

    // Verify the update was successful
    var returnedUser = JsonConvert.DeserializeObject<User>(response.Content);
    Assert.That(returnedUser.Name, Is.EqualTo("John Updated"));
}
```

DELETE Requests

```
[Test]
public void DeleteUser_ReturnsSuccess()
{
    // Create request
    var request = new RestRequest($"api/users/1", Method.Delete);

    // Execute request
    var response = _client.Execute(request);
```

```
    // Assert
    Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.NoContent));

    // Verify user no longer exists
    var getRequest = new RestRequest($"api/users/1", Method.Get);
    var getResponse = _client.Execute(getRequest);
    Assert.That(getResponse.StatusCode, Is.EqualTo(HttpStatusCode.NotFound));
}
```

Validating API Responses

Status Code Validation

```
// Positive test case
Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK));

// Negative test case
Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.BadRequest));
```

JSON Response Validation

```
// Parse response
var result = JsonConvert.DeserializeObject<User>(response.Content);

// Validate properties
Assert.That(result.Id, Is.GreaterThan(0));
Assert.That(result.Name, Is.EqualTo("Expected Name"));
Assert.That(result.Email, Is.Not.Null);
Assert.That(result.Email, Does.Contain("@"));
```

Response Headers Validation

```
// Check content type
Assert.That(response.ContentType, Is.EqualTo("application/json"));

// Check custom headers
Assert.That(response.Headers.Any(h => h.Name == "X-Pagination-Total" &&
h.Value.ToString() == "100"));
```

Schema Validation

For schema validation, you can use packages like `NJsonSchema`:

```
// Install NJsonSchema
// Install-Package NJsonSchema

[Test]
public void ValidateResponseSchema()
{
    var request = new RestRequest("api/users/1", Method.Get);
    var response = _client.Execute(request);

    Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK));

    // Define schema
    var schema = NJsonSchema.JsonSchema.FromType<User>();

    // Validate against schema
    var errors = schema.Validate(response.Content);
    Assert.That(errors.Count, Is.EqualTo(0), "Response does not match expected
schema");
}
```

Working with Postman

Postman is a popular API client that enables you to design, test, and document APIs.

Key Postman Features

- **Collections**: Group related API requests
- **Environments**: Define variables for different contexts (dev, staging, prod)
- **Tests**: Write JavaScript tests for responses
- **Automation**: Run collections with Newman (CLI)
- **Documentation**: Auto-generate API documentation

Writing Tests in Postman

Postman uses JavaScript for writing tests. Here are some examples:

Status Code Validation

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

Response Body Validation

```
pm.test("Response has expected user data", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.name).to.eql("John Doe");
```

```
    pm.expect(jsonData.email).to.exist;
});
```

Response Headers Validation

```
pm.test("Content-Type header is present", function () {
    pm.response.to.have.header("Content-Type");
    pm.expect(pm.response.headers.get("Content-
Type")).to.include("application/json");
});
```

Postman Assertion Types

Postman provides a rich set of assertion capabilities using the Chai Assertion Library. Here are various ways to test API responses:

Response Status Assertions

```
// Status code assertions
pm.test("Status code is 200", function() {
    pm.response.to.have.status(200);
});

// Status code class assertions
pm.test("Successful POST request", function() {
    pm.response.to.be.success; // Code is in the 2xx range
    pm.response.to.be.ok;      // Code is 200
    pm.response.to.be.created; // Code is 201
});

// Error status assertions
pm.test("Error handling", function() {
    pm.response.to.be.error;           // Code is in the 4xx or 5xx range
    pm.response.to.be.clientError;     // Code is in the 4xx range
    pm.response.to.be.serverError;     // Code is in the 5xx range
    pm.response.to.be.badRequest;      // Code is 400
    pm.response.to.be.unauthorized;    // Code is 401
    pm.response.to.be.forbidden;       // Code is 403
    pm.response.to.be.notFound;        // Code is 404
});
```

JSON Data Type Assertions

```
pm.test("Data type validations", function() {
    var jsonData = pm.response.json();
```

```
    // Type checking
    pm.expect(jsonData.userId).to.be.a('number');
    pm.expect(jsonData.name).to.be.a('string');
    pm.expect(jsonData.isActive).to.be.a('boolean');
    pm.expect(jsonData.tags).to.be.an('array');
    pm.expect(jsonData.profile).to.be.an('object');

    // Array assertions
    pm.expect(jsonData.tags).to.include('important');
    pm.expect(jsonData.tags).to.have.members(['important', 'user']);
    pm.expect(jsonData.tags).to.have.lengthOf(3);
});
```

Value Comparison Assertions

```
pm.test("Value validations", function() {
    var jsonData = pm.response.json();

    // Equality checks
    pm.expect(jsonData.name).to.equal('John Doe');
    pm.expect(jsonData.name).to.eql('John Doe');

    // Numeric comparisons
    pm.expect(jsonData.age).to.be.above(18);
    pm.expect(jsonData.price).to.be.below(100);
    pm.expect(jsonData.quantity).to.be.within(1, 10);

    // String comparisons
    pm.expect(jsonData.email).to.include('@example.com');
    pm.expect(jsonData.id).to.match(/^USER_\d+$/);

    // Existence checks
    pm.expect(jsonData.name).to.exist;
    pm.expect(jsonData.deletedAt).to.be.null;
    pm.expect(jsonData.optional).to.be.undefined;
});
```

Response Time Assertions

```
pm.test("Response time checks", function() {
    pm.expect(pm.response.responseTime).to.be.below(200);
});
```

Schema Validation

```
pm.test("Schema validation", function() {
    var schema = {
        "type": "object",
        "required": ["id", "name", "email"],
        "properties": {
            "id": { "type": "integer" },
            "name": { "type": "string" },
            "email": { "type": "string", "format": "email" },
            "tags": {
                "type": "array",
                "items": { "type": "string" }
            }
        }
    };

    pm.expect(pm.response.json()).to.be.jsonSchema(schema);
});
```

Chaining Assertions

```
pm.test("Chained assertions", function() {
    var jsonData = pm.response.json();

    pm.expect(jsonData.users).to.be.an('array')
        .and.to.have.lengthOf.at.least(1)
        .and.to.have.deep.property('[0].name');
});
```

Using Variables in Postman Tests

```
// Setting variables
pm.test("Set user ID for later use", function() {
    var jsonData = pm.response.json();
    pm.environment.set("userId", jsonData.id);
    pm.globals.set("apiVersion", jsonData.version);
});

// Using variables
pm.test("Use stored variables", function() {
    var userId = pm.environment.get("userId");
    var apiVersion = pm.globals.get("apiVersion");

    pm.expect(pm.request.url.toString()).to.include(`/users/${userId}`);
});
```

Best Practices

1. **Isolation**: Each test should be independent and not rely on other tests
2. **Data Management**: Use proper test data setup and teardown
3. **Authentication**: Handle auth tokens properly in tests
4. **Error Handling**: Test both positive and negative scenarios
5. **Parameterization**: Use data-driven testing for multiple test cases
6. **Environment Configuration**: Use config files for environment-specific settings
7. **Logging**: Implement proper logging for easier debugging
8. **Assertions**: Use descriptive assertions for better error messages
9. **Response Validation**: Always validate more than just status codes

Resources

- RestSharp Documentation
- Postman Learning Center
- NUnit Documentation
- RESTful API Design Best Practices
- JSON Schema Validation

This guide is meant as a reference. Adapt the examples to your specific API and testing requirements.