ASP.NET Boilerplate

Articles Tutorials

Filter...

latest (v7.1)

Edit on GitHub

In this document

Introduction

Wikipedia: "*An audit trail (also called audit log) is a security-relevant chronological record, set of records, and/or destination and source of records that provide documentary evidence of the sequence of activities that have affected at any time a specific operation, procedure, or event*".

ASP.NET Boilerplate provides the infrastructure to automatically log all interactions within the application. It can record intended method calls with caller info and arguments.

Basically, the saved fields are: Related **tenant id**, caller **user id**, called **service name** (the class of the called method), called **method name**, execution **parameters** (serialized into JSON), **execution time**, execution **duration** (in milliseconds), the client's **IP address**, the client's **computer name** and the **exception** (if the method throws an exception).

With this information, we not just know who did the operation, but we can also measure the **performance** of the application and observe the **exceptions** thrown. Furthermore, you can get **statistics** about the usage of your application.

The auditing system uses **IAbpSession** to get the current UserId and TenantId.

The Application Service, MVC Controller, Web API and ASP.NET Core methods are automatically audited by default.

About IAuditingStore

The auditing system uses **IAuditingStore** to save audit information. While you can implement it in your own way, it's fully implemented in the **Module Zero** project. If you don't implement it, SimpleLogAuditingStore is used and it writes audit information to the log.

Configuration

To configure auditing, you can use the **Configuration.Auditing** property in your module's PreInitialize method. Auditing is **enabled by default**. You can disable it as shown below:

Copy

```
public class MyModule : AbpModule
{
    public override void PreInitialize()
    {
        Configuration.Auditing.IsEnabled = false;
    }

    //...
}
```

Here are the auditing configuration properties:

- **IsEnabled**: Used to enable/disable the auditing system completely. Default: **true**.
- **IsEnabledForAnonymousUsers**: If this is set to true, audit logs are saved for users that are not logged in to the system. Default: **false**.
- **Selectors**: Used to select other classes to save audit logs.
- **SaveReturnValues**: Used to enable/disable to save return values. Default: **false**.
- **IgnoredTypes**: Used to ignore defined types.

Selectors is a list of predicates to select other types of classes that save audit logs. A selector has a unique **name** and a **predicate**. The only **default** selector in this list is used to select **application service classes**. It's defined as shown below:

Copy

```
Configuration.Auditing.Selectors.Add(
    new NamedTypeSelector(
        "Abp.ApplicationServices",
        type => typeof (IApplicationService).IsAssignableFrom(type)
    )
);
```

You can add your selectors in your module's PreInitialize method. You can also remove the selector above by name if you don't want to save audit logs for application services. This is why it has a unique name (Use simple LINQ to find the selector in Selectors and remove it if you want).

Note: In addition to the standard audit configuration, MVC and ASP.NET Core modules define configurations to enable/disable audit logging for actions.

Enable/Disable by attributes

While you can select auditing classes by configuration, you can use the **Audited** and **DisableAuditing** attributes for a single **class** or an individual **method**. Example:

Copy

```
[Audited]
public class MyClass
{
    public void MyMethod1(int a)
    {
        //...
    }

    [DisableAuditing]
    public void MyMethod2(string b)
    {
        //...
    }

    public void MyMethod3(int a, int b)
    {
        //...
    }
}
```

All methods of MyClass are audited except MyMethod2 since it's explicitly disabled. The Audited attribute can be used to save audits for the desired method.

DisableAuditing can also be used for a single **property of a DTO**. Thus, you can **hide sensitive data** in audit logs, such as passwords for example.

Notes

- A method must be **public** in order to be saved in audit logs. Private and protected methods are ignored.
- A method must be **virtual** if it's called over class reference. This is not needed if it's injected using its interface (like injecting the IPersonService interface to use the PersonService class). This is needed since ASP.NET Boilerplate uses dynamic proxying and interception. This is not true for **MVC** Controller actions. They may not be virtual.