

Exception Handling

Exception Constructs

Almost all language platforms offer a construct of exception or equivalent to handle error scenarios. The underlying platform, used libraries or the authored code can "throw" exceptions to initiate an error flow. Some of the advantages of using exceptions are -

1. Abstract different kind of errors
2. Breaks the control flow from different code structures
3. Navigate the call stack till the right catch block is identified
4. Automatic collection of call stack
5. Define different error handling flows thru multiple catch blocks
6. Define finally block to cleanup resources

Here is some guidance on exception handling in .Net

[C# Exception fundamentals](#)

[Handling exceptions in .Net](#)

Custom Exceptions

Although the platform offers numerous types of exceptions, often we need custom defined exceptions to arrive at an optimal low level design for error handling. The advantages of using custom exceptions are -

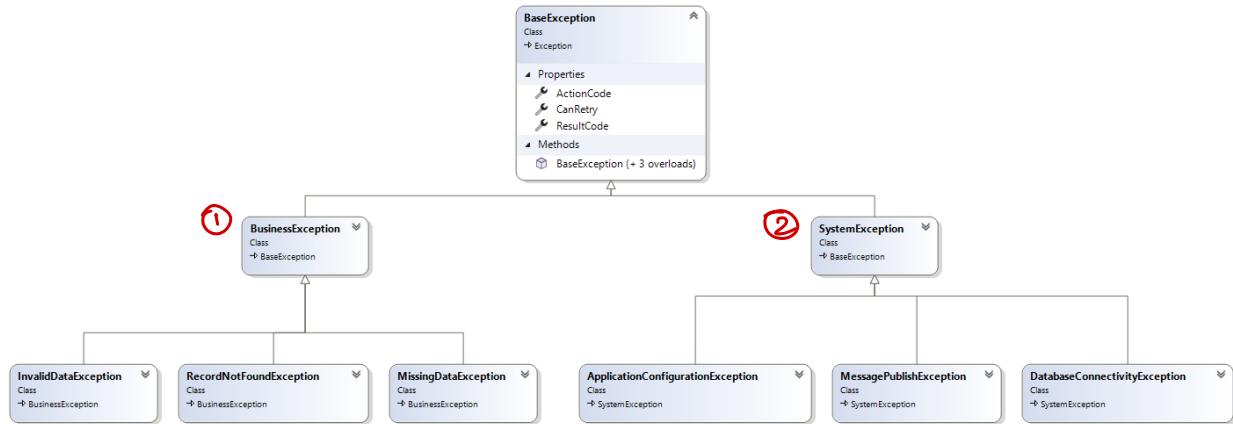
1. Define exceptions specific to business domain of the requirement. E.g. InvalidCustomerException
2. Wrap system/platform exceptions to define more generic system exception so that overall code base is more tech stack agnostic. E.g DatabaseWriteException which wraps MongoWriteException.
3. Enrich the exception with more information about the code flow of the error.
4. Enrich the exception with more information about the data context of the error. E.g. RecordId in property in DatabaseWriteException which carries the Id of the record failed to update.
5. Define custom error message which is more business user friendly or support team friendly.

Custom Exception Hierarchy

Below diagram shows a sample hierarchy of custom exceptions.

1. It defines a BaseException class which derives from System.Exception class and parent of all custom exceptions. BaseException also has additional properties for ActionCode and ResultCode. ActionCode represents the "flow" in which the error happened. ResultCode represents the exact error that happened. These additional properties help in defining different error handling flows in the catch blocks.
2. Defines a number of System exceptions which derive from SystemException class. They will address all the errors generated by the technical aspects of the code. Like connectivity, read, write, buffer overflow etc
3. Defines a number of Business exceptions which derive from BusinessException class. They will address all the errors generated by

the business aspects of the code. Like data validations, duplicate rows.



Error Details in API Response

When an error occurs in an API, it has to be rendered as response with all the necessary fields. There can be custom response schema drafted for these purposes. But one of the popular formats is the problem detail structure -

Problem details

There are inbuilt problem details middleware library built in ASP.Net core. For further details refer to below link

[Problem details service in ASP.Net core](#)

Last update: August 22, 2024