

# Currying Logs

---

This article is a tangent on Currying Databases.

*A little background:* Our student portal was riddled with bugs. Since we use a monolithic system like ASP.Net Classic, any tiny issue could cause our entire page to crash. The codebase is riddled with sloppy code handed down from person to person through 5 generations of programmers who formerly held the position. These people had no interest in cleaning up the code or making it coherent. To make matters worse, the vendor themselves had bugs that they did not fix.

Faculties and students would encounter bug sometimes and technicians could not replicate the issue. At times there would be bugs that students knew, but no one reported it. These are some of the embarrassing situations were common-place on the student portal.

To prevent an all-out-page-crash, a RESTful approach was used (when possible). Breaking **gigantic** operations into smaller units and wrapping each of those into a try-catch statements. Important operations that could fail should be inside a try-catch statements (e.g. database operations). When catching the error, supply as much information as possible such as variables and a simple and concise description of the error.

Currying is a great technique can be used to suite these purposes.

## How?

Currying allows us to string together expressions and execute them.

This is particularly useful when we need to supply important information inputs into a function.

For instance, this a simple API route that searches for a user. This route is reserved only for a small group of administrators.

```
try
{
    var users = PortalUserQueries.SearchPortalUser(q);

    if (ReferenceEquals(users, null))
        return "no data".ToResult(2).Resp();

    return users.ToResult(1).Resp();
}
catch(Exception ex)
{
    ex.Error("[Get Portal User] Error when trying to search for users via query.")
        .Add("q", q)
        .Add("qSearchPortalUser", PortalUserQueries.qSearchPortalUser)
        .Ok();
}
```

```
        return ex.Handle();
    }
```

(full code listed below)

We expect an array of rows in the form of `Dictionary<string, object>[]`. This is mimicking a JSON-like object and can be returned directly to the front-end. We always expect a null case in which there are no results.

The database operation here is `PortalUserQueries.SearchPortalUser(q)`, so we wrap it inside a try-catch clause. When an error occurs, we log the error via an extension method using our Curry log library. `ex.Error` begins the currying operation, we can supply additional information via the `Add` function. The `Ok` function commits the curry expression to our logging database. `ex.Handle()` is another extension method for Exception-based classes, returning a graceful message to the user in Release mode and reports an error message to us in Debug mode.

When an error occurs, we can see the error (exception object), the original query, and the input that was provided. If it was a query error, we could run the query with the input that caused the error to see what happens. Otherwise, we can check the exception object to view the stack trace.

Here's an example of a database call to get the Audit Records of a specific form entry.

```
try
{
    var auds = AuditQueries.GetAuditRows(_recid, _formid);

    if (ReferenceEquals(null, auds))
        "no data".ToResult(2).Resp();

    return auds.ToResult(1).Resp();
}
catch (Exception ex)
{
    ex.Error("[PZ Courses Audit] There was an error when trying to get audit
courses.")
        .Add("formid", formid)
        .Add("recid", recid)
        .Add("_formid (converted)", _formid.ToString())
        .Add("_recid (converted)", _recid.ToString())
        .Ok();

    return ex.Handle();
}
```

(full code listed below)

Here's an example of a database call to get the templates for forms.

```

try
{
    var templ = FormTemplateQueries.GetCourseFormTemplate(id);

    if (ReferenceEquals(templ, null))
        return "no data".ToResult(2).Resp();

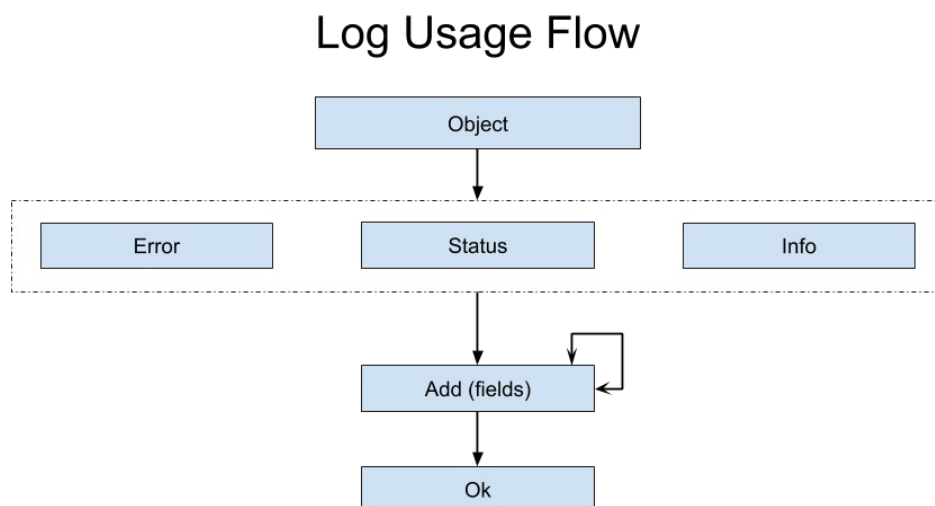
    return templ.ToResult(1).Resp();
}
catch (Exception ex)
{
    ex.Error("[PZCourses GetCourseFormTemplate] Error when trying to get
course form template.")
        .Add("formid", formid.ToString())
        .Add("qGetTemplate", FormTemplateQueries.qGetTemplate)
        .Ok();

    return ex.Handle();
}

```

(full code listed below)

## Usage Flow



## What else?

Logging done correctly could eliminate hours or days of troubleshooting. A good logging system could allow the administrators to anticipate potential issues before a huge one occurs. Another useful aspect of logging is it can help provide information and trace when necessary.

As an extra step, I created a service that would aggregate a list of errors within the past 24 hours at the end of the day. Looking at the daily report enough times, a pattern begins to emerge. When a problem occurs, I can

respond to it before the users complain. Sometimes, the report can do checks or display patterns that are unusual and this can help anticipate possible issues.

## Full Code

### Search Portal User Complete

```
public class PortalUserController : ApiController
{
    public HttpResponseMessage Post(string q)
    {
        if (PortalUser.Current.IsGuest)
            return new HttpResponseMessage(HttpStatusCode.Unauthorized);

        if (!PortalUser.Current.HasRole("Registrar")
            && !PortalUser.Current.IsSiteAdmin)
            return new HttpResponseMessage(HttpStatusCode.Forbidden);

        if(String.IsNullOrEmpty(q))
            return new HttpResponseMessage(HttpStatusCode.NotFound);

        try
        {
            var users = PortalUserQueries.SearchPortalUser(q);

            if (ReferenceEquals(users, null))
                return "no data".ToResult(2).Resp();

            return users.ToResult(1).Resp();
        }
        catch(Exception ex)
        {
            ex.Error("[Get Portal User] Error when trying to search for users
via query.")
                .Add("q", q)
                .Add("qSearchPortalUser", PortalUserQueries.qSearchPortalUser)
                .Ok();

            return ex.Handle();
        }
    }
}

public static class PortalUserQueries
{
    public const string qSearchPortalUser =
        @"select (FirstName + ' ' + LastName) fullname, substring(hostid, 4, 20)
        cxid, email from fwk_user"
```

```

        where FirstName like '%' + @name + '%' or LastName like '%' + @name +
        '%' or HostID like '%' + @cxid + '%'; ";

    public static Dictionary<string, object>[] SearchPortalUser(string q)
    {
        return Db.Jics
            .Cmd(qSearchPortalUser)
            .Param("@name", q)
            .Param("@cxid", q)
            .Rows();
    }
}

```

## Get Audit Records Complete

```

public class PZCoursesGetAuditRecsController : ApiController
{
    public HttpResponseMessage Get(string formid, string recid)
    {
        if (PortalUser.Current.IsGuest)
            return new HttpResponseMessage(HttpStatusCode.Unauthorized);

        if (!PortalUser.Current.HasRole("Registrar")
            && !PortalUser.Current.HasRole("Faculty")
            && !PortalUser.Current.IsSiteAdmin)
            return new HttpResponseMessage(HttpStatusCode.Forbidden);

        if (!int.TryParse(formid, out var _formid)
            || !int.TryParse(recid, out var _recid))
            return "bad input".ToResult(0).Resp();

        try
        {
            var auds = AuditQueries.GetAuditRows(_recid, _formid);

            if (ReferenceEquals(null, auds))
                "no data".ToResult(2).Resp();

            return auds.ToResult(1).Resp();
        }
        catch (Exception ex)
        {
            ex.Error("[PZ Courses Audit] There was an error when trying to get
            audit courses.")
                .Add("formid", formid)
                .Add("recid", recid)
                .Add("_formid (converted)", _formid.ToString())
                .Add("_recid (converted)", _recid.ToString())
        }
    }
}

```

```

        .Ok();

        return ex.Handle();
    }

}

}

public static class AuditQueries
{
    public static Dictionary<string, object>[] GetAuditRows(int recid, int
formid)
    {
        return Db.Forms
            .Proc("CourseForm_spGetAuditRecordsByRecId")
            .Param("@recid", recid)
            .Param("@formid", formid)
            .Rows();
    }
}

```

## Get Form Template Records Complete

```

public class GetCourseFormTemplateController : ApiController
{
    public HttpResponseMessage Get(string formid)
    {
        if (PortalUser.Current.IsGuest)
            return new HttpResponseMessage(HttpStatusCode.Unauthorized);

        if (!PortalUser.Current.HasRole("Registrar")
            && !PortalUser.Current.HasRole("Faculty")
            && !PortalUser.Current.IsSiteAdmin)
            return new HttpResponseMessage(HttpStatusCode.Forbidden);

        if (!int.TryParse(formid, out var id))
            return new HttpResponseMessage(HttpStatusCode.NotFound);

        try
        {
            var templ = FormTemplateQueries.GetCourseFormTemplate(id);

            if (ReferenceEquals(templ, null))
                return "no data".ToResult(2).Resp();

            return templ.ToResult(1).Resp();
        }
        catch (Exception ex)

```

```

        {
            ex.Error("[PZCourses GetCourseFormTemplate] Error when trying to
get course form template.")
            .Add("formid", formid.ToString())
            .Add("qGetTemplate", FormTemplateQueries.qGetTemplate)
            .Ok();

            return ex.Handle();
        }
    }

}

public static class FormTemplateQueries
{
    public const string qGetTemplate = "select * from _form_templates where id
= @id and category like 'Course Forms%'";

    public static Dictionary<string, object> GetCourseFormTemplate(int formid)
    {
        return Db.Forms
            .Cmd(qGetTemplate)
            .Param("@id", formid)
            .Row();
    }
}

```