Optimize Your Controller Layer

Matt Honeycutt
http://trycatchfail.com
mbhoneycutt@gmail.com

Why?

- Eliminate Errors
- Reduce Duplication
- Provide Consistency
- Improve Productivity

Outline

- Simplifying tedious mapping code
- Convention-based mapping configuration
- Eliminating magic strings from your controllers
- Standardizing feedback
- Creating a controller super-type
- Eliminating common code with action filters
- Before and After

Demos

Mapping	• Manual Mapping • AutoMapper Basics • Conventional Configuration
Eliminating Strings	• The String Problem • MVC Futures • Strongly-Typed Redirects
Standardizing Feedback	• Bootstrap Alerts • Decorating Action Results
Building a Controller Supertype	• Layer Supertypes • Custom Result Methods
Action Filters (again)	• Select List Woes • Conventional Action Filters

Demos

Mapping	• Manual Mapping • AutoMapper Basics • Conventional Configuration
Eliminating Strings	• The String Problem • MVC Futures • Strongly-Typed Redirects
Standardizing Feedback	• Bootstrap Alerts • Decorating Action Results
Building a Controller Supertype	• Layer Supertypes • Custom Result Methods
Action Filters (again)	• Select List Woes • Conventional Action Filters

Demos

Mapping

- Manual Mapping
- AutoMapper Basics
- Conventional Configuration

Eliminating Strings

- The String Problem
- MVC Futures
- Strongly-Typed Redirects

Standardizing Feedback

- Bootstrap Alerts
- Decorating Action Results

Building a Controller Supertype

- Layer Supertypes
- Custom Result Methods

Action Filters (again)

- Select List Woes
- Conventional Action Filters

The Decorator Pattern

AlertDecoratorResult : ActionResult

```
public AlertDecoratorResult(ActionResult innerResult)
{
        InnerResult = innerResult;
}

public override ExecuteResult(ControllerContext context)
{
        …

        InnerResult.ExecuteResult(context)
}
```

“*Design Patterns Library*” course on Pluralsight - http://goo.gl/UqFylb

Demos

Mapping

- Manual Mapping
- AutoMapper Basics
- Conventional Configuration

Eliminating Strings

- The String Problem
- MVC Futures
- Strongly-Typed Redirects

Standardizing Feedback

- Bootstrap Alerts
- Decorating Action Results

Building a Controller Supertype

- Layer Supertypes
- Custom Result Methods

Action Filters (again)

- Select List Woes
- Conventional Action Filters

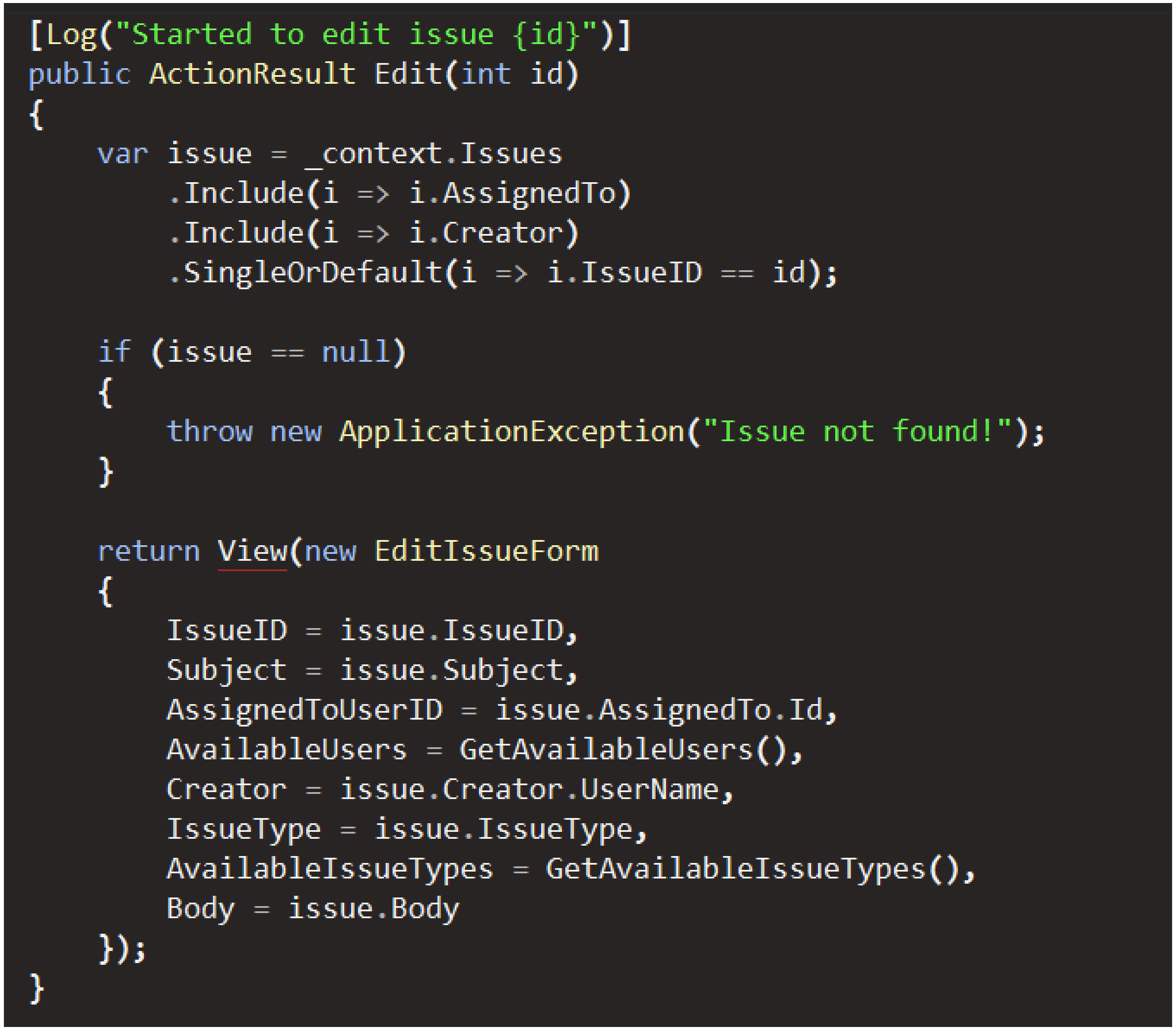
Demos

Mapping	• Manual Mapping • AutoMapper Basics • Conventional Configuration
Eliminating Strings	• The String Problem • MVC Futures • Strongly-Typed Redirects
Standardizing Feedback	• Bootstrap Alerts • Decorating Action Results
Building a Controller Supertype	• Layer Supertypes • Custom Result Methods
Action Filters (again)	• Select List Woes • Conventional Action Filters

Summary

- ✓ AutoMapper
- ✓ Flexible Mapping Conventions
- ✓ MVC Futures and Magic Strings
- ✓ Consistent Feedback UX
- ✓ Controller Super-Types
- ✓ Pass Options to a Select List

Before & After

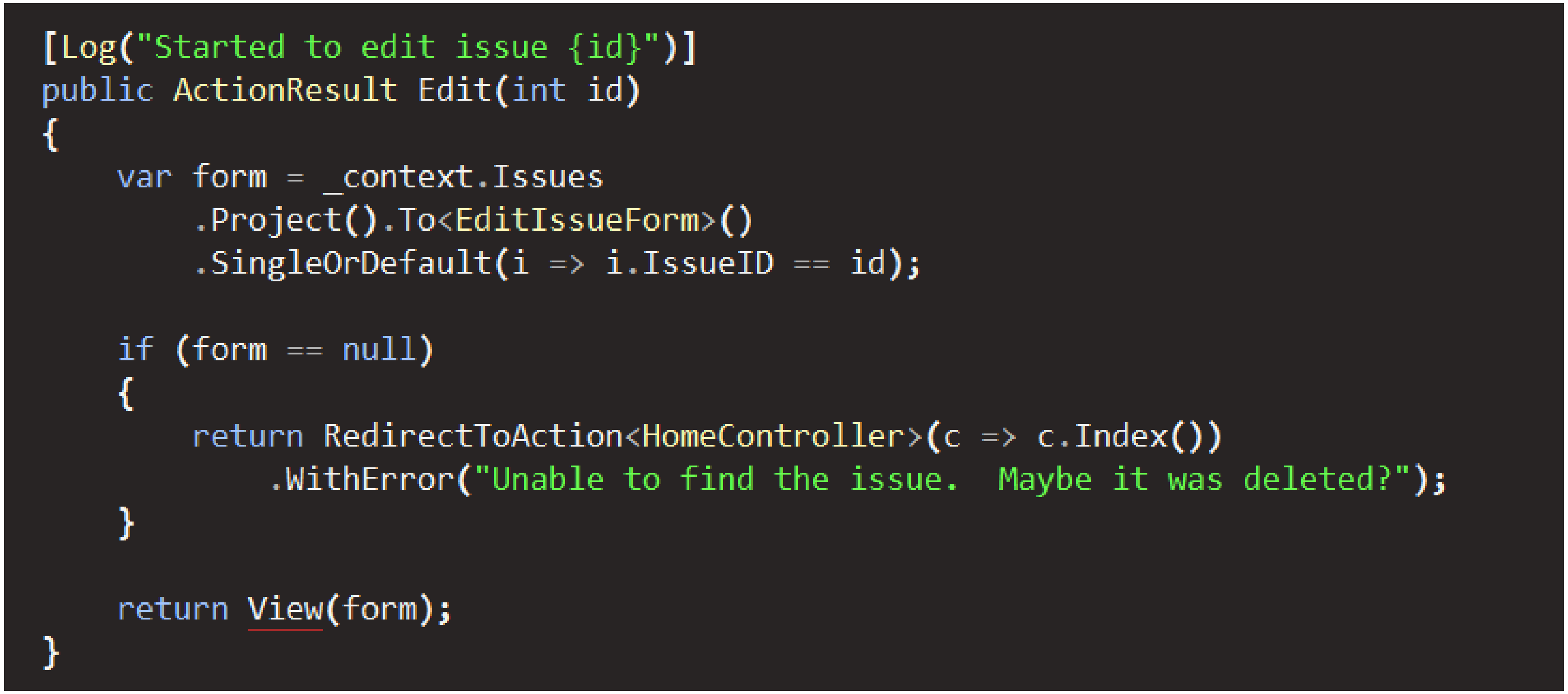


```
[Log("Started to edit issue {id}")]
public ActionResult Edit(int id)
{
    var issue = _context.Issues
        .Include(i => i.AssignedTo)
        .Include(i => i.Creator)
        .SingleOrDefault(i => i.IssueID == id);

    if (issue == null)
    {
        throw new ApplicationException("Issue not found!");
    }

    return View(new EditIssueForm
    {
        IssueID = issue.IssueID,
        Subject = issue.Subject,
        AssignedToUserID = issue.AssignedTo.Id,
        AvailableUsers = GetAvailableUsers(),
        Creator = issue.Creator.UserName,
        IssueType = issue.IssueType,
        AvailableIssueTypes = GetAvailableIssueTypes(),
        Body = issue.Body
    });
}
```

Before & After



```
[Log("Started to edit issue {id}")]
public ActionResult Edit(int id)
{
    var form = _context.Issues
        .Project().To<EditIssueForm>()
        .SingleOrDefault(i => i.IssueID == id);

    if (form == null)
    {
        return RedirectToAction<HomeController>(c => c.Index())
            .WithError("Unable to find the issue.  Maybe it was deleted?");
    }

    return View(form);
}
```

Up next: Streamline your view