PREVENTING INJECTIONS

.NET MVC SANITIZATION & VALIDATION BASICS

WHAT WE'RE COVERING

What is Sanitisation and Validation

Why they are done

.NET MVC's Tools to provide Sanitisation and Validation

Other methods

WHAT IS VALIDATION?

Data validation is the process of ensuring the accuracy and quality of data before processing it.

Different types of validation can be performed depending on our application needs.

Data validation is a form of data sanitisation.

WHAT IS VALIDATION?

Data validation can be used to make sure that our data meets the rules for what type and format the data should be in.

This could be things like making sure the data is a number, email or phone number format.

It could also be the process of checking that the data also comes from a valid source which is accepted by the system.

DATA INPUT SANITISATION

Input sanitisation checks data once it has been entered and removes anything that might be potentially dangerous.

It looks for any sequences of text/symbols or commands that may have been inserted within the data and removes or substitutes it.

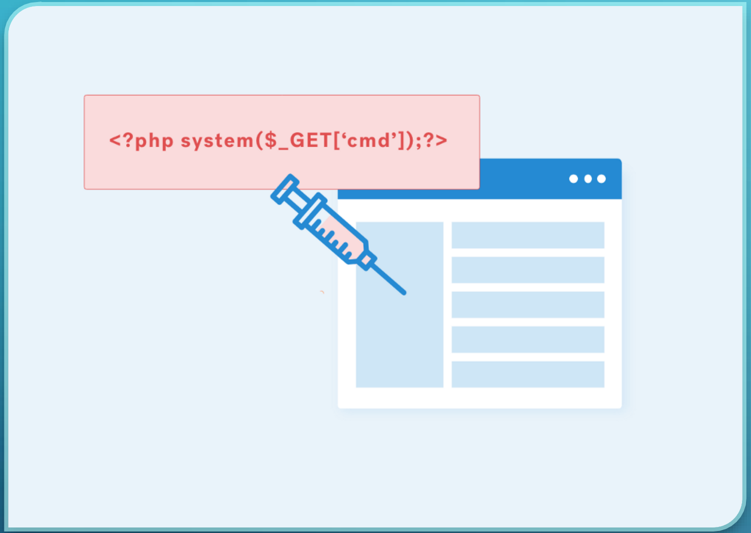
It's basically the process of cleaning your data inputs before they are used.

DATA INPUT SANITISATION

Sanitisation is needed because sometimes an input can meet the requirements of what is considered a valid input, but still hold dangerous content inside it.

For example, your validation might just be checking for a string within a certain size. But the provided string might contain symbols or sequences that if converted to html, might insert a script into your webpage.

WHY DO WE USE THEM?



```
<?php system($_GET['cmd']);?>
```

Because of how some systems in programming work. Text inputs are often used and then injected to commands and queries to build code within these systems.

This can allow malicious operators to use the injected text/data to manipulate the command and change it to something that adds to or changes how the command works.

WHY DO WE USE THEM?

This is very common with systems like SQL and HTML because they generally use plain text which is then interpreted into commands or functionality.

It is during this conversion processing step where the commands can be re-written or modified.

WHY WE USE THEM

By adding data validation we can minimise peoples ability to insert these types of malicious inputs into our system by making sure the data conforms with the rules of our data.

Sanitisation then checks the input once it has been entered and looks for any particular sequences of characters that are intended to break and modify where the input is going to be used.

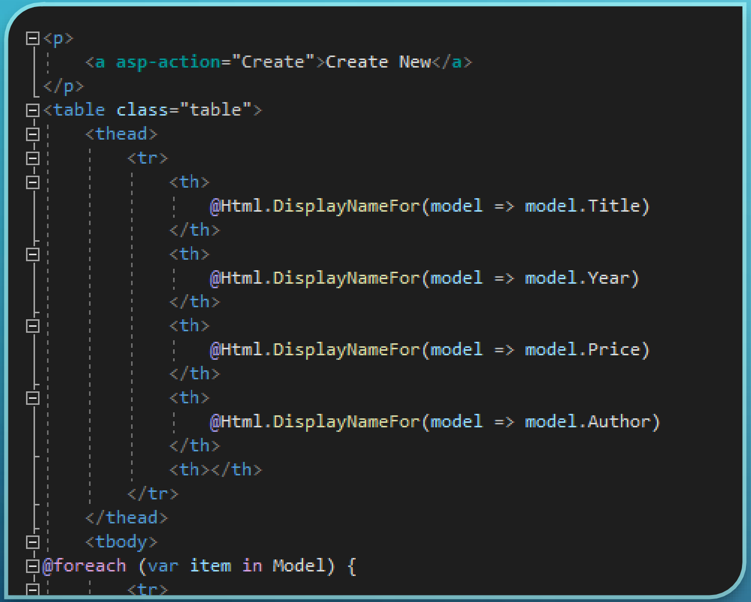
ASP.NET MVC TO THE RESCUE

Thankfully many modern development frameworks have a number of in-built tools to help with Validation and Sanitisation of data in order to prevent attacks.

Many of these are automatically included in your code without even needing to do anything yourself.

ASP.NET MVC

MVC VIEWS



```
<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Year)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Price)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Author)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
```

By default .NET MVC views do not use any methods that render HTML directly into their pages.

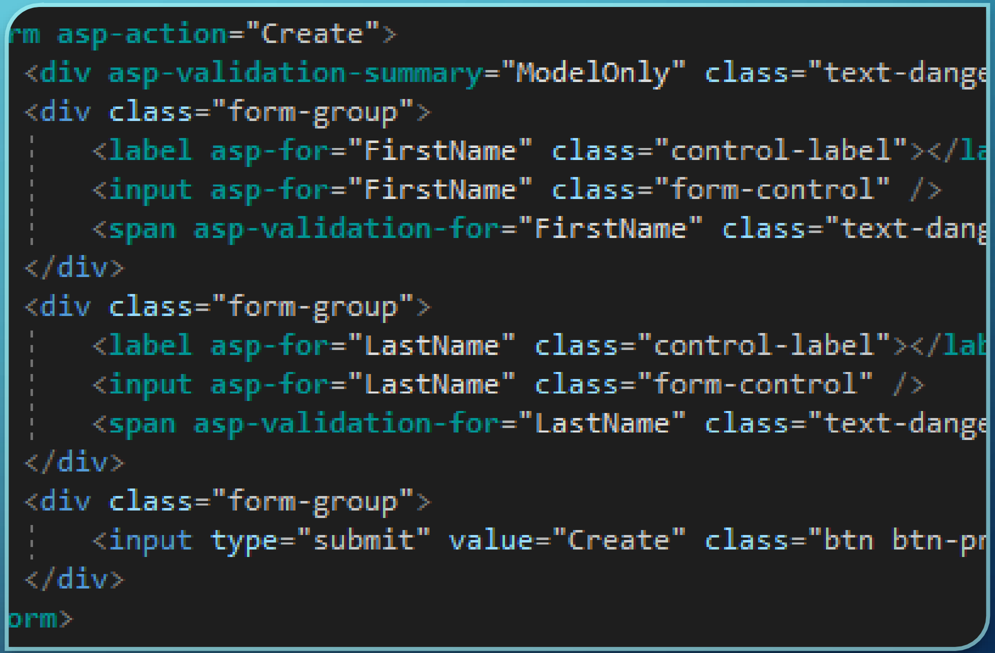
You can still allow this to occur by using the correct commands, but the default templates and scaffolding avoids using methods that let this occur.

This stops any text entered into the system from inserting commands into the view as it will only ever render the content as plain text.

ANTI FORGERY TOKENS

To help prevent CSRF(Cross-Site Request Forgery) attacks, ASP.NET MVC uses anti-forgery tokens, also called *request verification tokens*.

When the client requests an HTML page that contains a form. The server includes two tokens in the response. One token is sent as a cookie. The other is placed in a hidden form field. The tokens are generated randomly so that an adversary cannot guess the values.



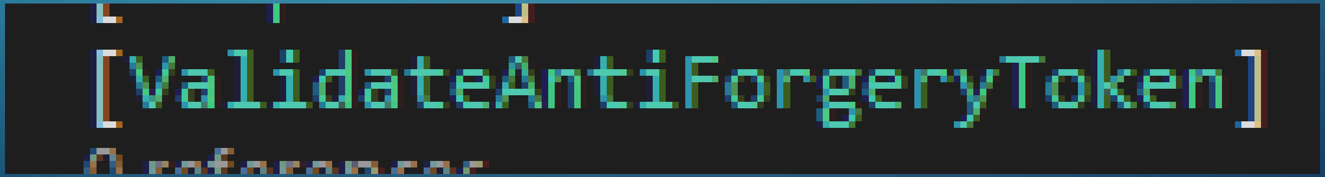
```
rm asp-action="Create">
 <div asp-validation-summary="ModelOnly" class="text-dange
 <div class="form-group">
     <label asp-for="FirstName" class="control-label"></la
     <input asp-for="FirstName" class="form-control" />
     <span asp-validation-for="FirstName" class="text-dang
 </div>
 <div class="form-group">
     <label asp-for="LastName" class="control-label"></lab
     <input asp-for="LastName" class="form-control" />
     <span asp-validation-for="LastName" class="text-dange
 </div>
 <div class="form-group">
     <input type="submit" value="Create" class="btn btn-pr
 </div>
orm>
```

ANTI FORGERY TOKENS

When the client submits the form, it must send both tokens back to the server. The client sends the cookie token as a cookie, and it sends the form token inside the form data. (A browser client automatically does this when the user submits the form.)

If a request does not include both tokens, the server disallows the request.

To ensure this check is done, all you need to do is include the following annotation above the HTTP Post action for that form.

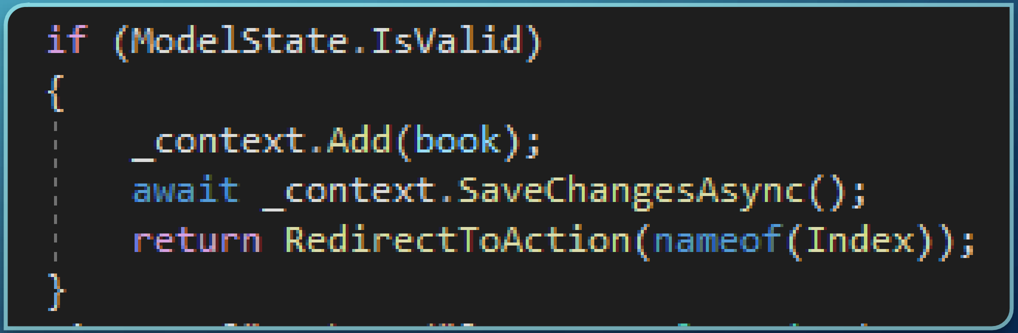


```
[ValidateAntiForgeryToken]
```

MODEL VALIDATION

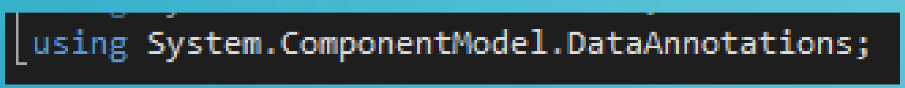
Whenever a form passes data to a data model in your controller, there is a Model State check which checks whether the data in the model conforms with the validation rules assigned to the data.

This will flag a TRUE or FALSE value in the model state which can be checked prior to performing and additional code.

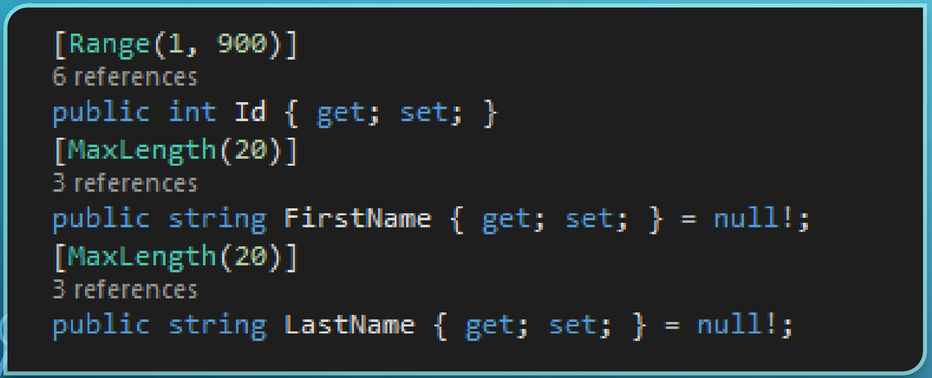


```
if (ModelState.IsValid)
{
    _context.Add(book);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}
```

MODEL VALIDATION



```
using System.ComponentModel.DataAnnotations;
```



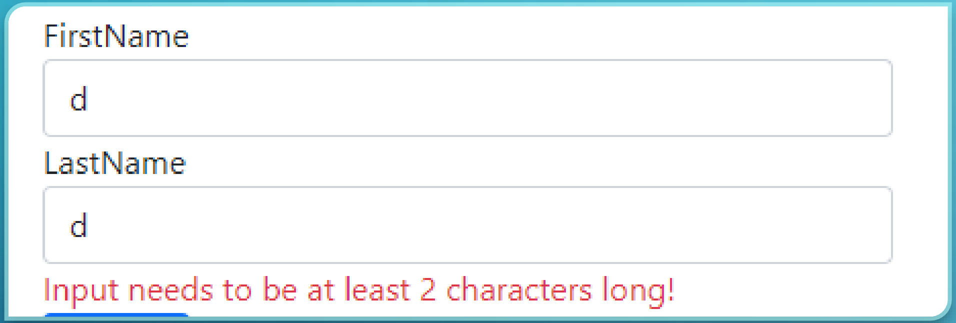
```
[Range(1, 900)]
6 references
public int Id { get; set; }
[MaxLength(20)]
3 references
public string FirstName { get; set; } = null!;
[MaxLength(20)]
3 references
public string LastName { get; set; } = null!;
```

Often the default rules for your model state is just to ensure the relevant fields have been provided with a value.

However, you can apply additional rules and constraints by using annotations above each of the model properties which can be used to specify various things such as data character lengths, types and formats.

Just add the namespace shown on the left, and you are ready to go.

MODEL VALIDATION


FirstName

d

LastName

d

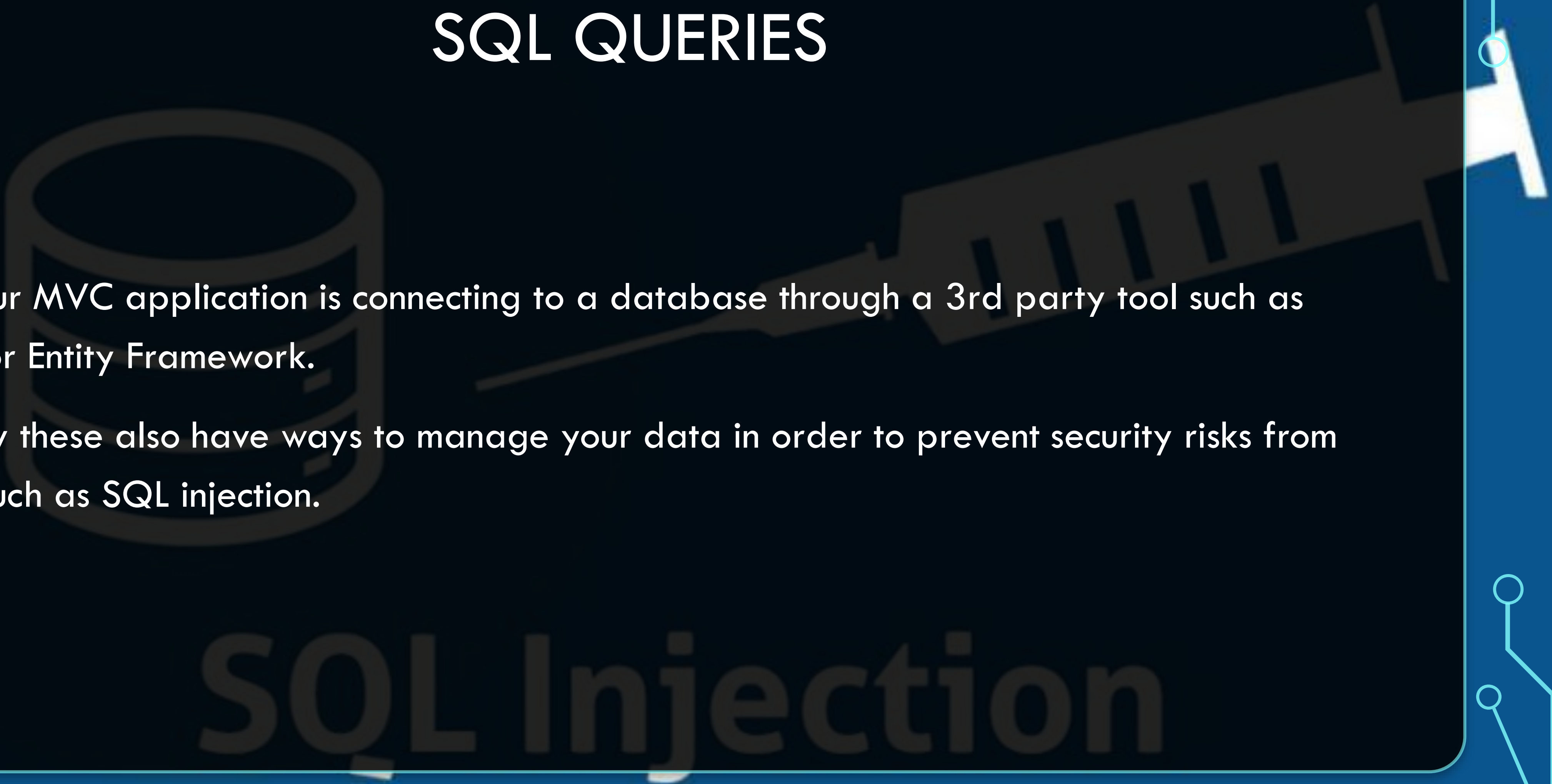
Input needs to be at least 2 characters long!

Not only do the annotations set the rules for acceptable data, they also apply changes to the views associated with the properties.

If you set a max length value for a string, the form will only allow the user to type that many characters into the input field. If you specify an input needs to be an email, it will war the user if they enter something that does not meet the rules for a valid email address.

Some rules even change the type of input field used.

SQL QUERIES



Often your MVC application is connecting to a database through a 3rd party tool such as Dapper or Entity Framework.

Thankfully these also have ways to manage your data in order to prevent security risks from systems such as SQL injection.

PARAMETERISED QUERIES

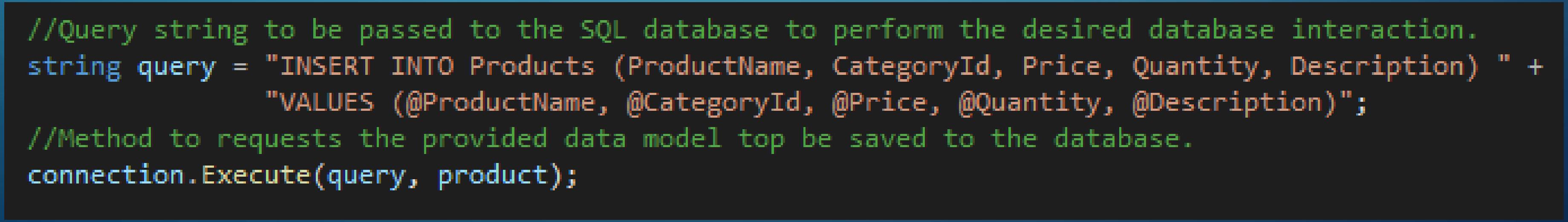
The most common way that ORM's such as Entity Framework or Dapper prevent SQL injection is by using parameterised queries.

This is done automatically by Entity Framework each time you make a request to your context class unless you are directly entering the SQL string manually to the request.

PARAMETERISED QUERIES

In Dapper we just write our queries to use parameters by using the @ symbol and parameter name within the queries and handing it a model to pull the data from.

When the query is handled by the database, the values are then inserted into the string in a safe manner which stopes them from being able to break the query or add to it.



```
//Query string to be passed to the SQL database to perform the desired database interaction.
string query = "INSERT INTO Products (ProductName, CategoryId, Price, Quantity, Description) " +
               "VALUES (@ProductName, @CategoryId, @Price, @Quantity, @Description)";
//Method to requests the provided data model top be saved to the database.
connection.Execute(query, product);
```

OTHER SYSTEMS

In most cases these processes will generally keep you safe and secure when managing data inputs in your system.

However, sometimes due to your requirements of the application, you need to bypass some of these systems to achieve your clients desired functionality.

In this case you may need to add additional or supplementary systems to your application. Thankfully there are a lot of great sanitisation libraries in the NuGet package manager which you can pass data inputs into and get them sanitised before you use them.

QUESTIONS?