WEB PROGRAMMING ASP.NET MVC CORE

Without validation, users could enter nonsense data or even submit an empty form. In an MVC application, you will typically apply validation to the domain model rather than in the user interface. This means that you define validation in one place, but it takes effect anywhere in the application that the model class is used.

MVC supports declarative validation rules defined with attributes from the `System.ComponentModel.DataAnnotations` namespace, meaning that validation constraints are expressed using the standard C# attribute features.

MVC automatically detects the attributes and uses them to validate data during the model-binding process.

ADDING VALIDATIONS

ADDING VALIDATIONS

```
using System.ComponentModel.DataAnnotations;

namespace PartyInvites.Models {
    public class GuestResponse {
        [Required(ErrorMessage = "Please enter your name")]
        public string Name { get; set; }

        [Required(ErrorMessage = "Please enter your phone number")]
        public string Phone { get; set; }

        [Required(ErrorMessage = "Please enter your email address")]
        public string Email { get; set; }

        public bool? WillAttend { get; set; }
    }
}
```

ADDING VALIDATION

If the `ModelState.IsValid` property returns false , then I know that there are validation errors. The object returned by the `ModelState` property provides details of each problem that has been encountered, but we don't need to get into that level of detail, because we can rely on a useful feature that automates the process of asking the user to address any problems by calling the View method without any parameters.

```
[HttpPost]
public ViewResult Rsvp(GuestResponse response) {
    if (ModelState.IsValid) {
        Repository.AddResponse(response);
        return View("Thanks", response);
    } else {
        // There are validation errors
        return View();
    }
}
```

ADDING VALIDATION

When MVC renders a view, Razor has access to the details of any validation errors associated with the request, and tag helpers can access the details to display validation errors to the user.

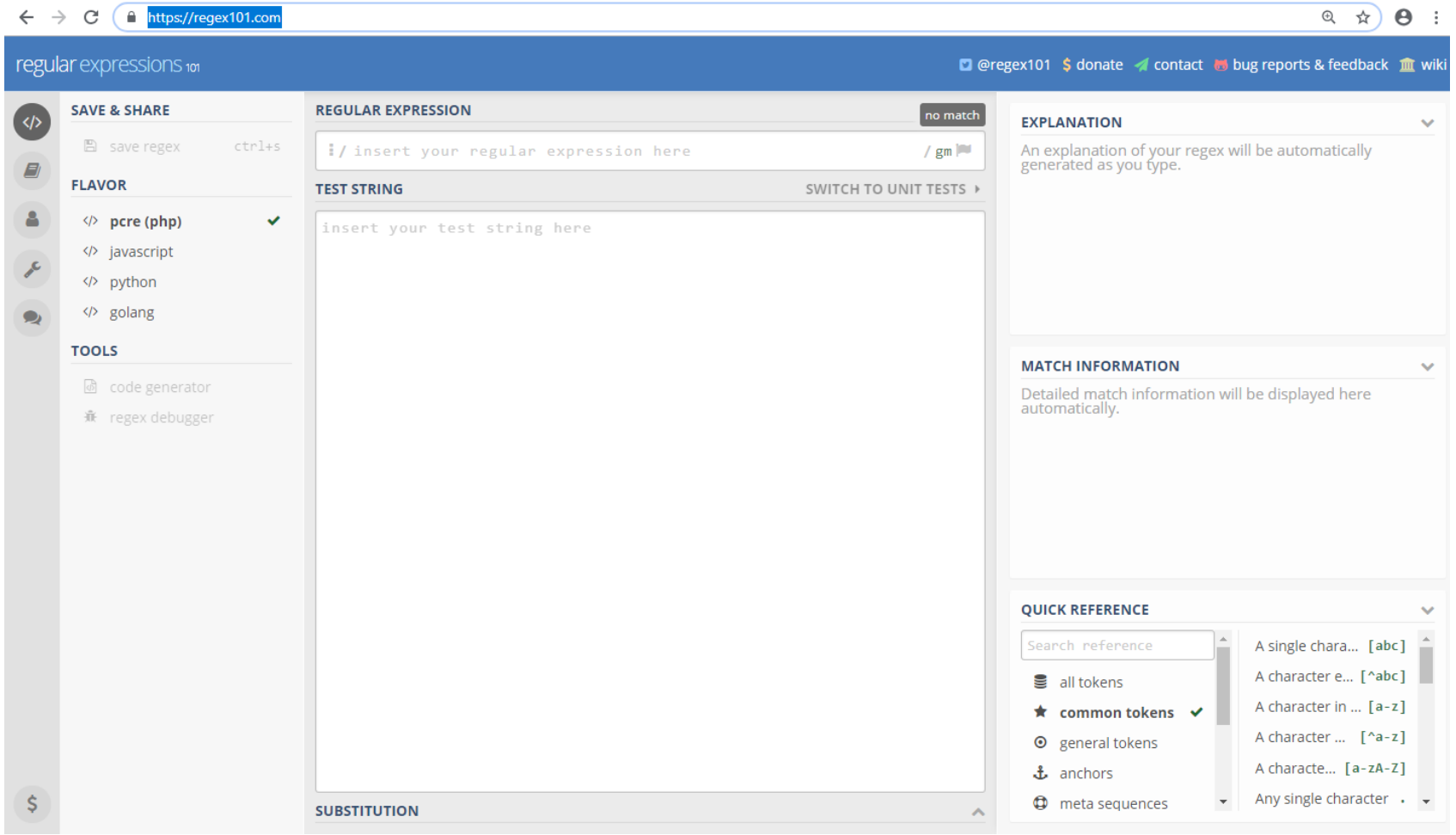
The `asp-validation-summary` attribute is applied to a div element, and it displays a list of validation errors when the view is rendered.

```
<form asp-action="Register" method="post">
    <!-- ... -->

    <div asp-validation-summary="All"></div>
</form>
```

REGULAR EXPRESSION VALIDATIONS

```
using System.ComponentModel.DataAnnotations;

namespace PartyInvites.Models {
    public class GuestResponse {
        [Required(ErrorMessage = "Please enter your name")]
        public string Name { get; set; }

        [Required(ErrorMessage = "Please enter your phone number")]
        public string Phone { get; set; }

        [Required(ErrorMessage = "Please enter your email address")]
        [RegularExpression(@"(\w+(\.\w+)*@[a-zA-Z_]+?\.[a-zA-Z]{2,6})",
        ErrorMessage = "Please enter a valid email address")]
        public string Email { get; set; }

        public bool? WillAttend { get; set; }
    }
}
```

REGULAR EXPRESSIONS



- https://regex101.com
- https://regexper.com/
- https://www.debuggex.com

REGULAR EXPRESSIONS – MATCH CHARACTERS

REGULAR EXPRESSION

/ abc

TEST STRING

abc
xptoabcabx
olaabc

REGULAR EXPRESSIONS – MATCH ANY CHARACTER

REGULAR EXPRESSION

```
/ .abc.
```

TEST STRING

```
abc
xptoabcabx
olaabc
```

REGULAR EXPRESSIONS – MATCH AT LEAST ONE QUANTIFIER

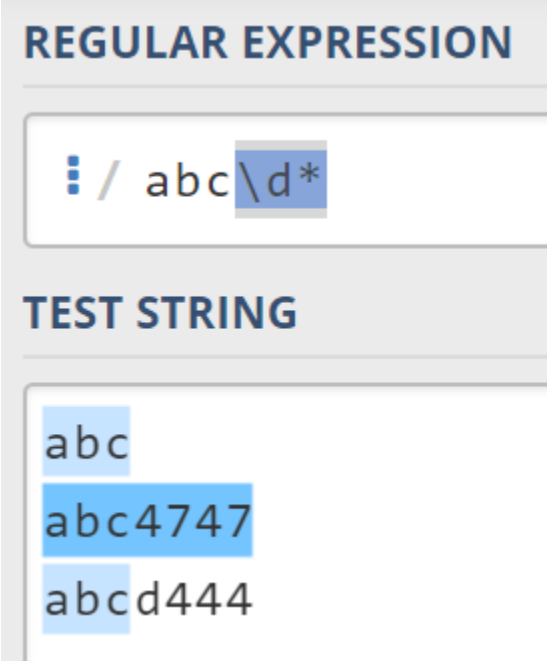
REGULAR EXPRESSION

```
/ .+abc.+
```

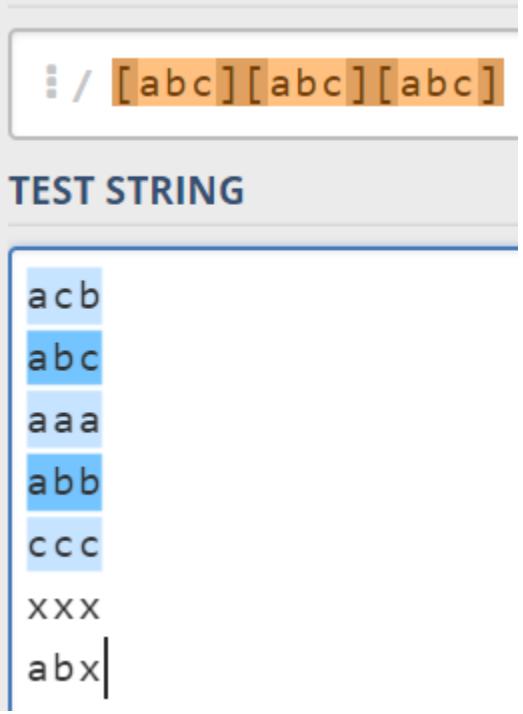
TEST STRING

```
abc
xptoabcabx
olaabc
abcola
```

REGULAR EXPRESSIONS – MATCH ZERO OR MORE QUANTIFIER



REGULAR EXPRESSIONS – GROUPS OF CHARS



REGULAR EXPRESSIONS – GROUPS OF CHARS

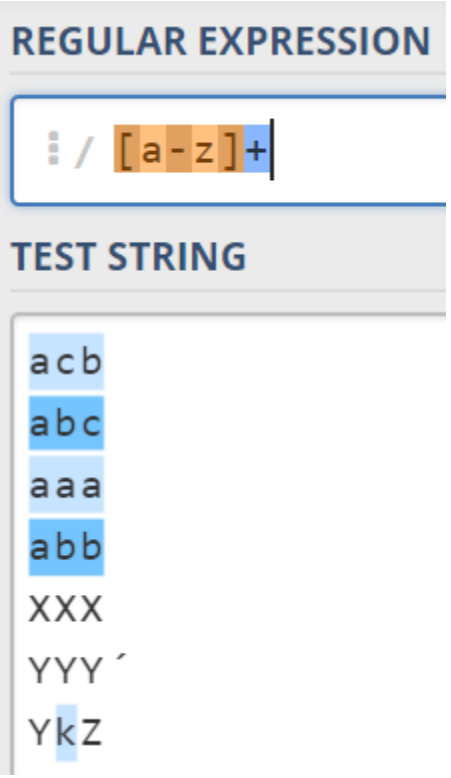
REGULAR EXPRESSION

/ [abc][abc][a-z]

TEST STRING

acb
abc
aaa
abb
ccc
xxx
abx

REGULAR EXPRESSIONS – GROUPS OF CHARS



REGULAR EXPRESSIONS – GROUPS OF CHARS

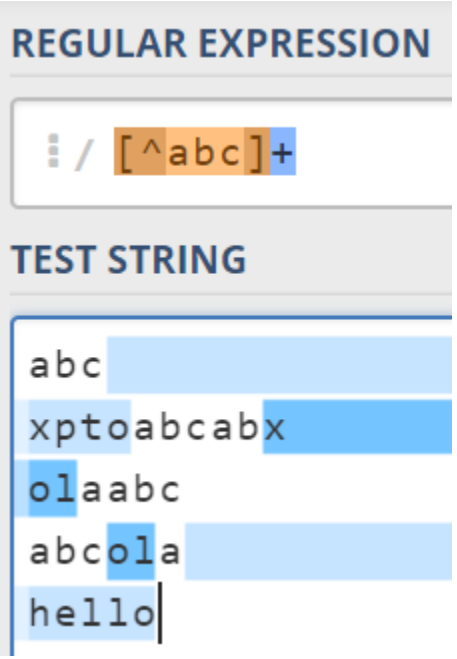
REGULAR EXPRESSION

```
/ [a-zA-Z0-9]+
```

TEST STRING

```
acb
abc
aaa
abb
XXX
YYY
YkZ
7A9
$%#
```

REGULAR EXPRESSIONS – NOT



REGULAR EXPRESSIONS – NUMERIC DIGITS

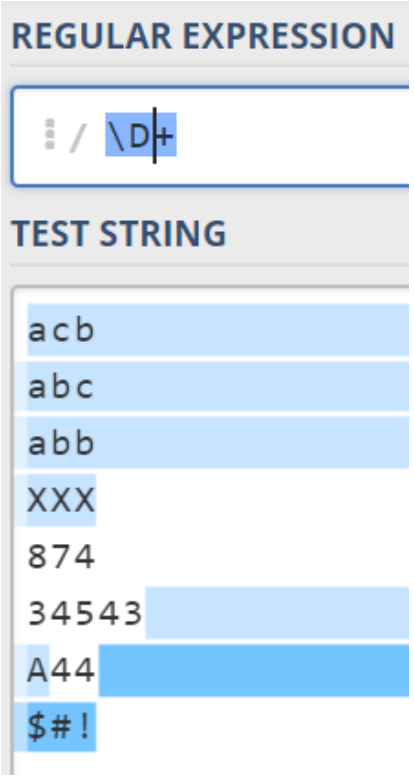
REGULAR EXPRESSION

/ \d+

TEST STRING

acb
abc
abb
XXX
874
34543
A44

REGULAR EXPRESSIONS – NON NUMERIC DIGITS



REGULAR EXPRESSIONS – SPACES

REGULAR EXPRESSION

/ \s+

TEST STRING

```
acb xpto

abc
abb
XXX
874
34543
A44
$#!
```

REGULAR EXPRESSIONS – NON SPACES

REGULAR EXPRESSION

```
/ \S+
```

TEST STRING

```
acb xpto

abc
abb
XXX
874
34543
A44
$#!
```

REGULAR EXPRESSIONS – WORDS

REGULAR EXPRESSION

```
/ \w+
```

TEST STRING

```
acb xpto

abc
abb___
XXX
874
34543
A44
$#!
```

REGULAR EXPRESSIONS – NON WORD

REGULAR EXPRESSION

/ \W+

TEST STRING

```
acb xpto

abc
abb___
XXX
874
34543
A44
$#!
```

REGULAR EXPRESSIONS – EXACTLY QUANTIFIER

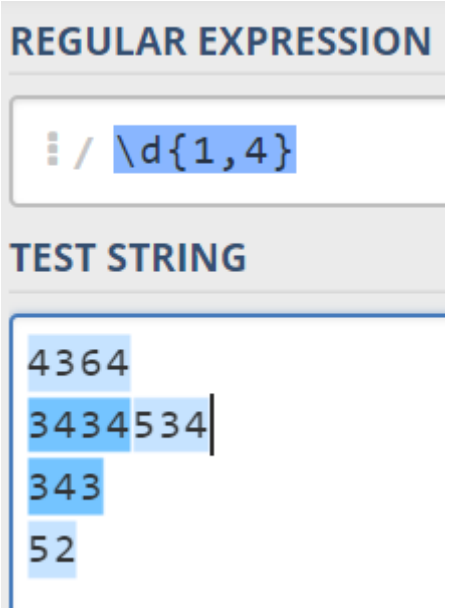
REGULAR EXPRESSION

```
/ \d{4}
```

TEST STRING

```
6300-559
3343-544
6300-75
6300-
6300
wewe
233a
```

REGULAR EXPRESSIONS – BETWEEN QUANTIFIER



REGULAR EXPRESSIONS – MINIMUM QUANTIFIER

REGULAR EXPRESSION

/ \d{2,}

TEST STRING

4364
3434534
343
52
1

REGULAR EXPRESSIONS – GROUPS AND OPTIONAL

REGULAR EXPRESSION

```
/ \d{4}(-\d{3})?
```

TEST STRING

```
6300-559
3343-544
6300-75
6300-
6300
wewe
233a
```

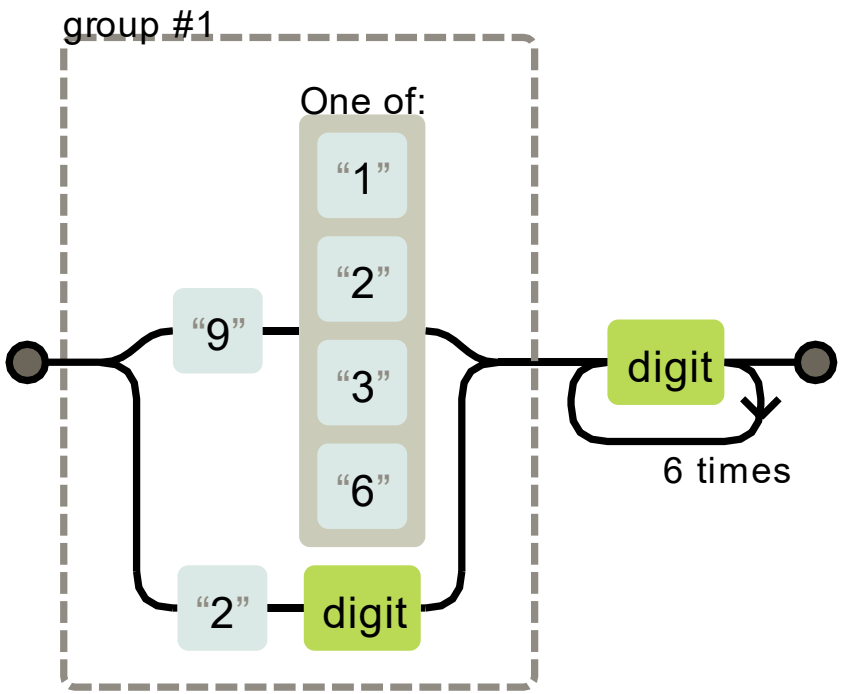
REGULAR EXPRESSIONS – OR

REGULAR EXPRESSION

```
/ (9[1236]|2\d)\d{7}
```

TEST STRING

```
913473484
271534553
943473484
843473484
3243237947
34443434
```



REGULAR EXPRESSIONS – EMAIL

REGULAR EXPRESSION

```
/ \w+(\.\w+)*@\w+(\.\w+)?
```

TEST STRING

```
noel@ipg.pt
noel_lopes_5@gmail.com
noel.lopes.xpto@gmail.com
joao.çoelho@gáail.com
noelipg
hello@
@ddsffssf
hello@ddhhd
abc@abc
abc@abc.com
```

EMAIL ADDRESS VALIDATION

```
public class GuestResponse {
    [Required(ErrorMessage ="Please enter your name")]
    public string Name { get; set; }

    [Required(ErrorMessage = "Please enter your phone number")]
    public string Phone { get; set; }

    [Required(ErrorMessage = "Please enter your email")]
    //[RegularExpression(@"(\w+(\.\w+)*@[a-zA-Z_]+?\.[a-zA-Z]{2,6})", ErrorMessage ="Invalid email")]
    [EmailAddress]
    public string Email { get; set; }

    public bool? WillAttend { get; set; }
}
```

STRING LENGTH VALIDATION

```
using System.ComponentModel.DataAnnotations;

namespace PartyInvites.Models {
    public class GuestResponse {
        [Required(ErrorMessage = "Please enter your name")]
        [StringLength(50, MinimumLength = 3)]
        public string Name { get; set; }

        // ...
    }
}
```

OTHER VALIDATIONS

HTTPS://DOCS.MICROSOFT.COM/EN-US/ASPNET/CORE/MVC/MODELS/VALIDATION?VIEW=ASPNETCORE-3.1

BUILT-IN VALIDATION ATTRIBUTES

- [CreditCard]: Validates that the property has a credit card format.
- [Compare]: Validates that two properties in a model match.
- [EmailAddress]: Validates that the property has an email format.
- [Phone]: Validates that the property has a telephone number format.
- [Range]: Validates that the property value falls within a specified range.
- [RegularExpression]: Validates that the property value matches a specified regular expression.
- [Required]: Validates that the field is not null. See [Required] attribute for details about this attribute's behavior.
- [StringLength]: Validates that a string property value doesn't exceed a specified length limit.
- [Url]: Validates that the property has a URL format.
- [Remote]: Validates input on the client by calling an action method on the server. See [Remote] attribute for details about this attribute's behavior.

HIGHLIGHTING INVALID FIELDS

- When there are invalid fields, the data entered is preserved and displayed again. This is another benefit of model binding, and it simplifies working with form data.

```
.field-validation-error {
    color: #f00;
}

.field-validation-valid {
    display: none;
}

.input-validation-error {
    border: 1px solid #f00;
    background-color: #fee;
}

.validation-summary-errors {
    font-weight: bold;
    color: #f00;
}

.validation-summary-valid {
    display: none;
}
```