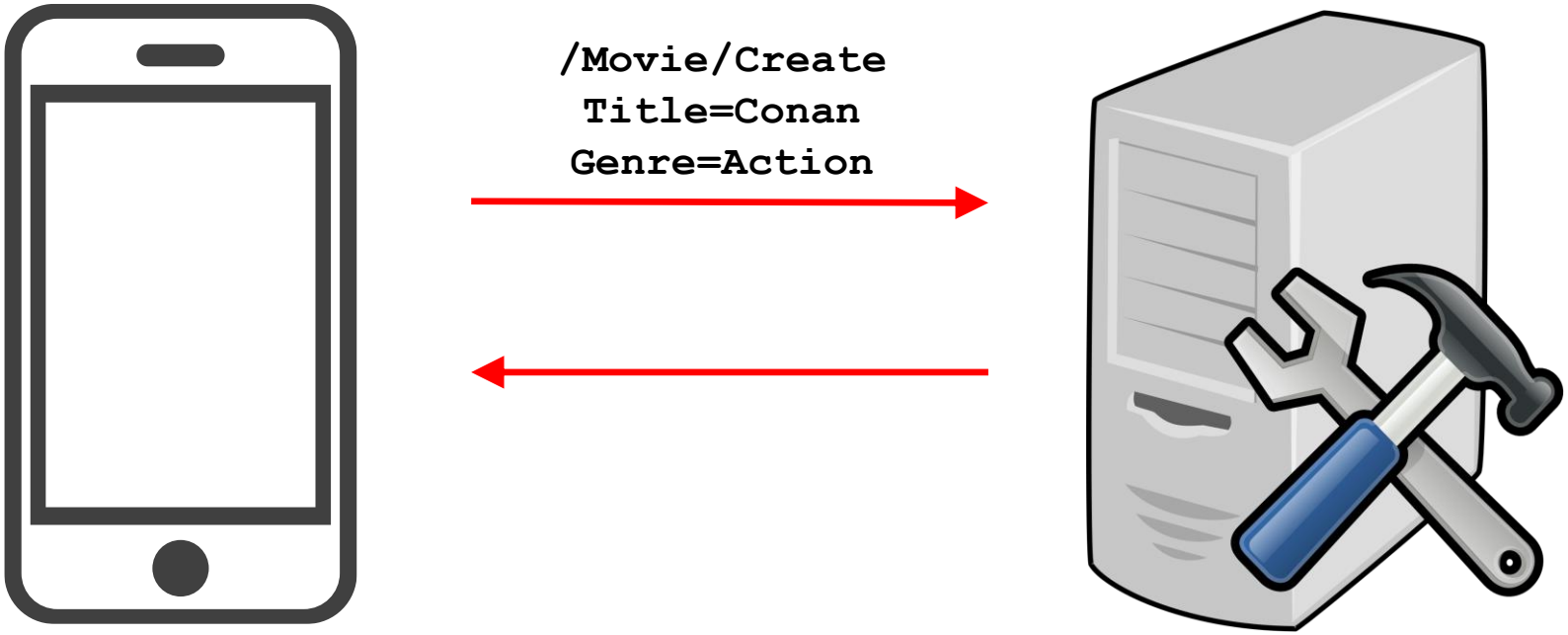
ASP.NET CORE

MODEL BINDING

issntt@nus.edu.sg

Problem



Client

Server

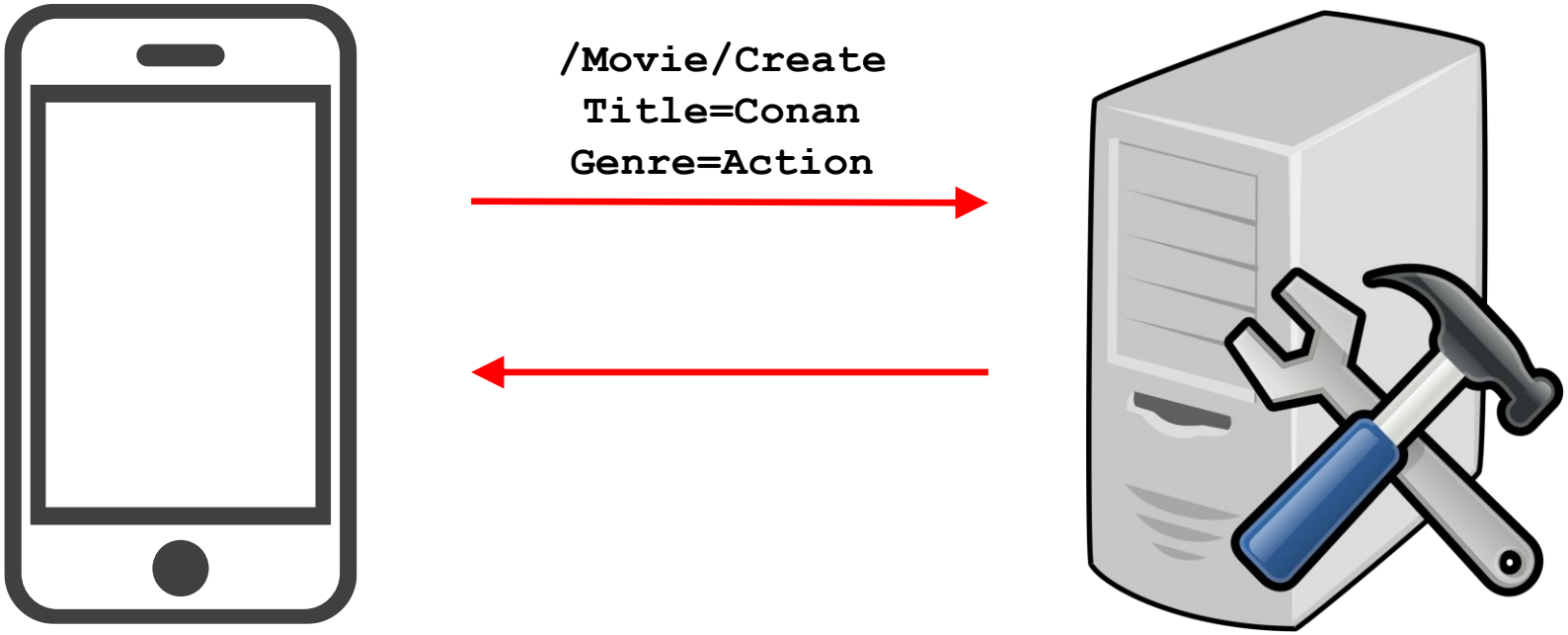
How can our **web apps retrieve data from clients**?

Objectives

At the end of this lesson, students will be able to

- Describe different options clients can use to send data to servers
- Describe how ASP.NET Core automatically extracts request data and uses them to create objects that are passed as parameters of action methods
- Design and implement parameters of action methods using Model Binding
- Differentiate among Binding Model, Application Model and View Model in the context of MVC

First Problem



Client

Server

How can **clients send data** to servers?

Topics

- **HTTP Request revisit**
  - **Query String**
  - **Method (Verb)**
  - **POST Request**
- Model Binding
- More details M in MVC

URL Anatomy

1 2 3 4 5

http://www.example.com:56563/Staff/View?type=Lecturer&sortBy=name

4. **Path**: specify the **resource path**
5. **Query string: extra parameters** sent to the server
   - A **list** of **key/value pairs** separated by the **& symbol**

HTTP Request

Following is a **sample request** when Browsers access http://localhost:52845/Staff/View?type=lecturer&sort=name

method path and string query

```
GET /Staff/View?type=lecturer&sort=name HTTP/1.1
Host: localhost:52845
User-Agent: Firefox/3.6.10
Accept: text/html,application/xhtml+xml
```

host

headers

Headers are in **key-value pairs,** and we can add our **custom headers**, too

https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#HTTP_Messages

How is each part of URL being used in the HTTP GET Request?

HTTP Request Method (Verb)

By HTTP design, clients are **expected** to use request methods to indicate the **desired actions** from the Server

Method (Verb)	Purpose
GET	To fetch resource(s)
POST	To save/create resource(s)
PUT	To replace/update existing resource(s)
DELETE	To delete resource(s)

Typically, clients **fetch** a resource (using GET) or **post** the value of an HTML form (using POST)

Developers can still use POST to fetch or resources, etc., but those are considered bad practices in web development today

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

A HTTP POST Request

Following is a possible request to **create a new** movie

method

```
POST /Movie/Create HTTP/1.1
Host: localhost:52845
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml

Title=Conan&Genre=Action&Price=1.99
```

headers

body

Which part of the request is used to send the movie's details?

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST

Question

How can we make Browsers send a POST Request?

```
<form action="/Movie/Create"
                    method="post">
  <input type="text" id="Title"
             name="Title" /><br/>
  <input type="text" id="Genre"
             name="Genre" /><br/>
  <input type="text" id="Price"
             name="Price" /><br/>
  <input type="submit"
      value="Create" class="btn" />
</form>
```

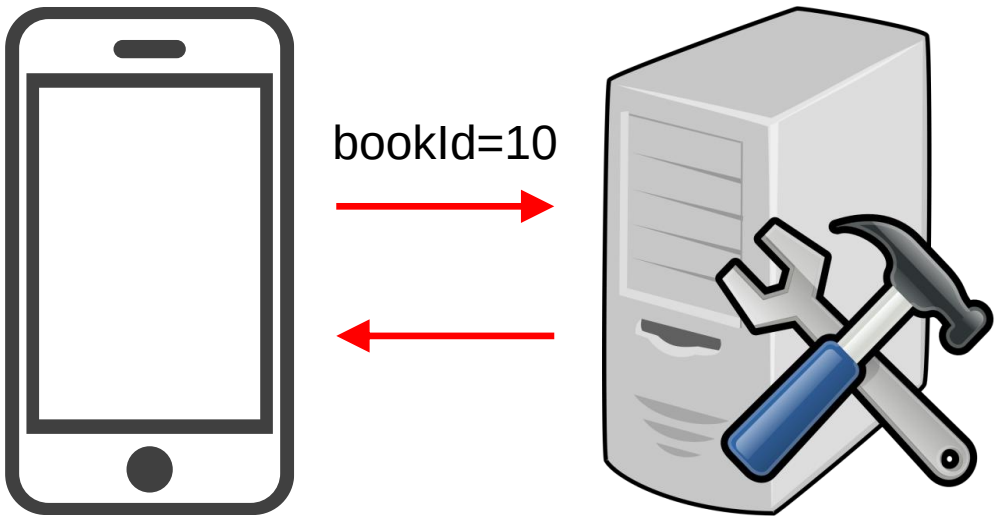
Conan
Action
1.99

Create

```
POST /Movie/Create HTTP/1.1
Host: localhost:52845
...

Title=Conan&Genre=Action&Price
=1.99
```

Question

A client needs to **send** a **book ID (e.g., 10)** to the server. Using HTTP Request, **how many options** can it use to send the data?



Second Problem

Client

3. Send a **request**

```
POST /Movie/Edit/10 HTTP/1.1
Host: localhost:52845
...

Title=Casablanca&Genre=Cla
    ssics
```

The request maps the action method, but **how** can **servers retrieve** the **data** from requests (and then **map** them to the **method parameters**)?

Server

1. Has **configured** some **routing**, for example

```
app.MapControllerRoute(
  name: "default",
  pattern:
   "{controller=Home}/{action=Index}/{id?}");
```

2. Has some **controller** with **action methods**, for example

```
public class MovieController
                    : Controller {
   public IActionResult Edit(
            int id, Movie movie) {
      // method implementation
      // need to use id and movie
   }
}
```

Topics

- HTTP Request revisit
- **Model Binding**
  - Binding Collections
  - Binding Complex Objects
  - Choosing a binding source
- More details M in MVC

Model Binding

Model Binding means **extracting request data** and using them to **create objects** that are passed as **parameters** of action methods

```
GET /Movie/Details/10 HTTP/1.1
Host: localhost:52845
...
```

```
public class MoviesController
                    : Controller
{
   public IActionResult
                Details(int id)
   {
      // method implementation
      // will use id
   }
}
```

The binding is done **automatically** by .NET Core MVC, but we need to **configure** properly

How Model Binding happens?

By **default**, .NET Core MVC uses **three main** different **binding sources** when creating binding models, **in order**

1
Form values

In the body of an HTTP request when a form is sent using a POST

2
Route parameter values

Obtained from **URL segments** or through **default values** after a route is matched

3
Query string values

Passed at the end of the URL

In total, there are 5 sources, read more

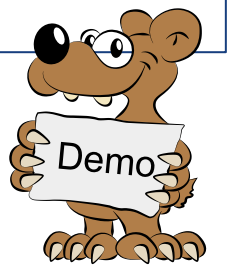
3. Query Strings

Data can be bound from a query string, where **query** and **method parameter names** must **match**

```
app.MapControllerRoute(
  name: "default",
  pattern: "{controller=Home}/{action=Index}/{id?}");
```

```
GET
/Home/Say?msg=H
iDipSA&times=3
```

```
public class HomeController : Controller {
  public IActionResult Say(
            string msg, int times) {
    for (int i = 0; i < times; i++) {
      Debug.WriteLine(msg);
    }
    // method implementation
  }
}
```



2. Route Parameter Values

Data can be obtained from **route parameter values** after a route is **matched**

```
app.MapControllerRoute(
  name: "default",
  pattern: "{controller=Home}/{action=Index}/{id?}");
```

```
GET
/Movie/Details/10
```

```
public class MovieController
                    : Controller {
   public IActionResult
                Details(int id) {
      Debug.WriteLine(id);
      // method implementation
   }
}
```

1. Form Values

Data can be obtained from the **body** of an HTTP request when a form is sent using a POST

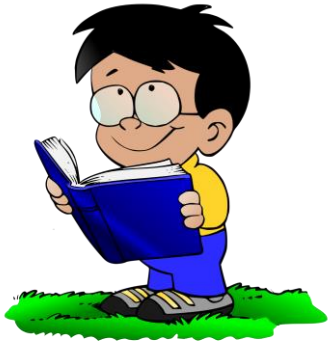
```
POST
/Movie/Create

Title=Conan&Genre=Action&Price=1.99
```

```
public IActionResult Create(
        string title,
        string genre,
        double price) {
   Debug.WriteLine(title);
   Debug.WriteLine(genre);
   Debug.WriteLine(price);
   // method implementation
}
```

Quiz – Currency Converter

Consider a method that

- Users provide a **quantity** in one **input currency**
- Our app converts it to another **output currency**



Quiz – Currency Converter

The routing template is given below. For each case, what are the **values** of each of parameters in the action method?

`{controller}/{action}/{currencyIn}/{currencyOut}`.

#	URL path	HTTP body
1	`CurrencyConverter/Convert/SGD/USD`	
2	`CurrencyConverter/Convert/SGD/USD?quantity=100`	
3	`CurrencyConverter/Convert/SGD/USD?quantity=100`	quantity=50
4	`CurrencyConverter/Convert/SGD/USD?quantity=100&currencyOut=JPY`	quantity=50&currencyIn=EUR

```
public class
    CurrencyConverterController
                        : Controller {
  public IActionResult Convert (
      string currencyIn,
      string currencyOut,
      int quantity)
  {
     // method implementation
  }
}
```

For better **code maintenance**, except special purposes such as authentication, try to put all data in **one source** only

Next

Users want to view **a number of** movie items, e.g., ID 2, 4 and 10

Client

Send a list of movie IDs

Server

Has some **controller** with **action methods**, for example

```
public class MoviesController : Controller {
   public IActionResult Details(
                        List<long> IDList)
   {
      // method implementation
      // need to use IDList
   }
}
```

How can the client send the **list** for a **successful binding**?

Topics

- HTTP Request revisit
- **Model Binding**
  - **Binding Collections**
  - **Binding Complex Objects**
  - Choosing a binding source
- More details M in MVC

Binding Collections

Client sends **collection data** to server using the **same key** for **different values**, or using **index**

GET /Movie/Details?**IDList=2&IDList=4&IDList=10**

OR

GET /Movie/Details?**IDList[0]=2&IDList[1]=4&IDList[2]=10**

```
public class MovieController : Controller
{
   public IActionResult Details(
                List<long> IDList)
   {
      foreach (long id in IDList)
      {
         Debug.WriteLine(id);
      }
      // method implementation
   }
}
```

Next

Consider this scenario

```
public class EmployeeController : Controller {
   [HttpPost]
   public IActionResult Create(
                string firstName,
                string lastName,
                string email,
                string dob,
                string designation) {
      // method implementation
   }
}
```

Too **many** parameters

Primitive binding is useful, but can .NET Core bind the **whole object**?

Binding Complex Objects

A complex object can be bound by (1) **declaring** a **Binding Model class** and (2) **using** it in action **parameter lists**

```
public class 1 EmployeeBindingModel
{
   public a EmployeeBindingModel() {}
b public string FirstName { get; b set; }
   public string LastName { get; set; }
   public string Email { get; set; }
   public string Dob { get; set; }
   public string Designation { get; set; }
}
```

```
public class EmployeeController : Controller {
   [HttpPost]
   public IActionResult Create(
         2 EmployeeBindingModel emp) {
      // method implementation
   }
}
```

Binding Complex Objects

.NET Core loops through **all properties** of the binding model and attempts to find **name-value pairs** that **matches**


```
public class EmployeeBindingModel {
   public EmployeeBindingModel() {}
   public string FirstName { get;   set; }
   public string LastName { get; set; }
   public string Email { get; set; }
   public string Dob { get; set; }
   public string Designation { get; set; }
}
```

```
POST
/Employee/Create

FirstName=Potte
r&LastName=Harr
y&Email=hp@mom.
gov&Dob=31 Jul
1980&Designati
on=Magician
```

```
public class EmployeeController : Controller {
   [HttpPost]
   public IActionResult Create(
            EmployeeBindingModel emp) {
      Debug.WriteLine(@"FirstName: {0}, LastName: {1},
         Email: {2}, Dob: {3}, Designation: {4}",
         emp.FirstName, emp.LastName, emp.Email,
         emp.Dob, emp.Designation);
      // method implementation
   }
}
```

We can **explicitly** tell **which properties** to bind, see Reading list

Binding Complex Objects

Self study

For **multiple** complex **parameters** that contain the **same properties**, pass data **prefixed** with the **names** of the models

```
POST
/Employee/CreateTwo

emp1.FirstName=Potter&
emp1.LastName=Harry&emp1
.Email=hp@mom.gov&emp1.D
ob=31 Jul
1980&emp1.Designation=Ma
gician&emp2.FirstName=
Granger&emp2.LastName=He
rmione&emp2.Email=hg@mom
.gov&emp2.Dob=19 Sep
1979&emp2.Designation=Mi
nister
```

```
public class EmployeeController : Controller {
  [HttpPost]
  public IActionResult CreateTwo(
          EmployeeBindingModel emp1,
          EmployeeBindingModel emp2) {
    Debug.WriteLine(@"FirstName: {0}, LastName: {1},
      Email: {2}, Dob: {3}, Designation: {4}",
      emp1.FirstName, emp1.LastName, emp1.Email,
      emp1.Dob, emp1.Designation);

    Debug.WriteLine(@"FirstName: {0}, LastName: {1},
      Email: {2}, Dob: {3}, Designation: {4}",
      emp2.FirstName, emp2.LastName, emp2.Email,
      emp2.Dob, emp2.Designation);

    // method implementation
  }
}
```

```
FirstName: Potter, LastName: Harry,
Email: hp@mom.gov, Dob: 31 Jul 1980,
Designation: Magician
FirstName: Granger, LastName Hermione,
Email: hg@mom.gov, Dob: 19 Sep 1979,
Designation: Minister
```

With this technique, we can bind complex **hierarchical** models

Next

- So far, .NET Core **automatically** bind data from **forms**, **route parameters** and **query string**
- But clients can also use HTTP **Request header** to send data to servers

```
POST /Staff/View
Host: localhost:52845
User-Agent: Firefox/3.6.10
Accept: text/html
MyCustomKey: my custom
   value

Title=Casablanca&Genre=Classics
```

How can servers bind those data?

Using header is **more common** when developing **Web API**

Topics

- HTTP Request revisit
- **Model Binding**
  - Binding Collections
  - Binding Complex Objects
  - **Choosing a binding source (Self Study)**
- More details M in MVC

Choosing a binding source

Self study

Add *[FromHeader]* **before** each action method **parameter** to specify the **request header** as the **binding source**

```
GET /api/TodoItems HTTP/1.1
Host: localhost:59136
clientId: 77a06fd447b4
```

```
[Route("api/[controller]")]
public class TodoItemsController : ControllerBase {
   [HttpGet]
   public IEnumerable<TodoItem> GetTodoItems(
                                [FromHeader] string clientId) {
      Debug.WriteLine(clientId);
      // method implementation
   }
}
```

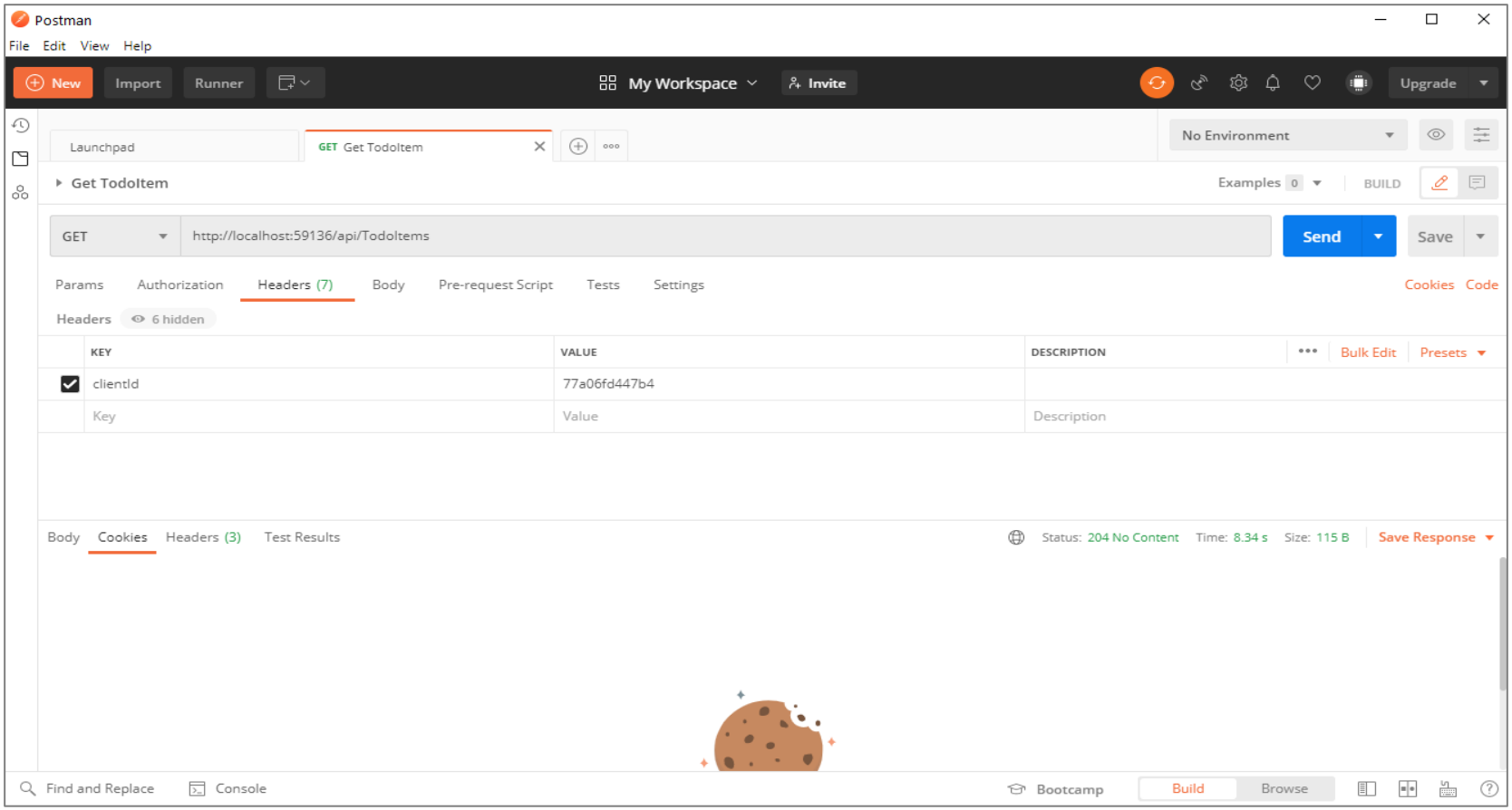
How can we generate HTTP request with our custom headers for testing?

```
77a06fd447b4
```

Postman

Self study

Postman is a **client tool** that easily **generates HTTP Requests, analyzes HTTP Responses** and much more



Intro to Postman https://www.youtube.com/watch?v=7E60ZttwIpY

Choosing a binding source

Self study

In fact, we can also **specify** any **binding source** of some action method **parameter** in the same manner

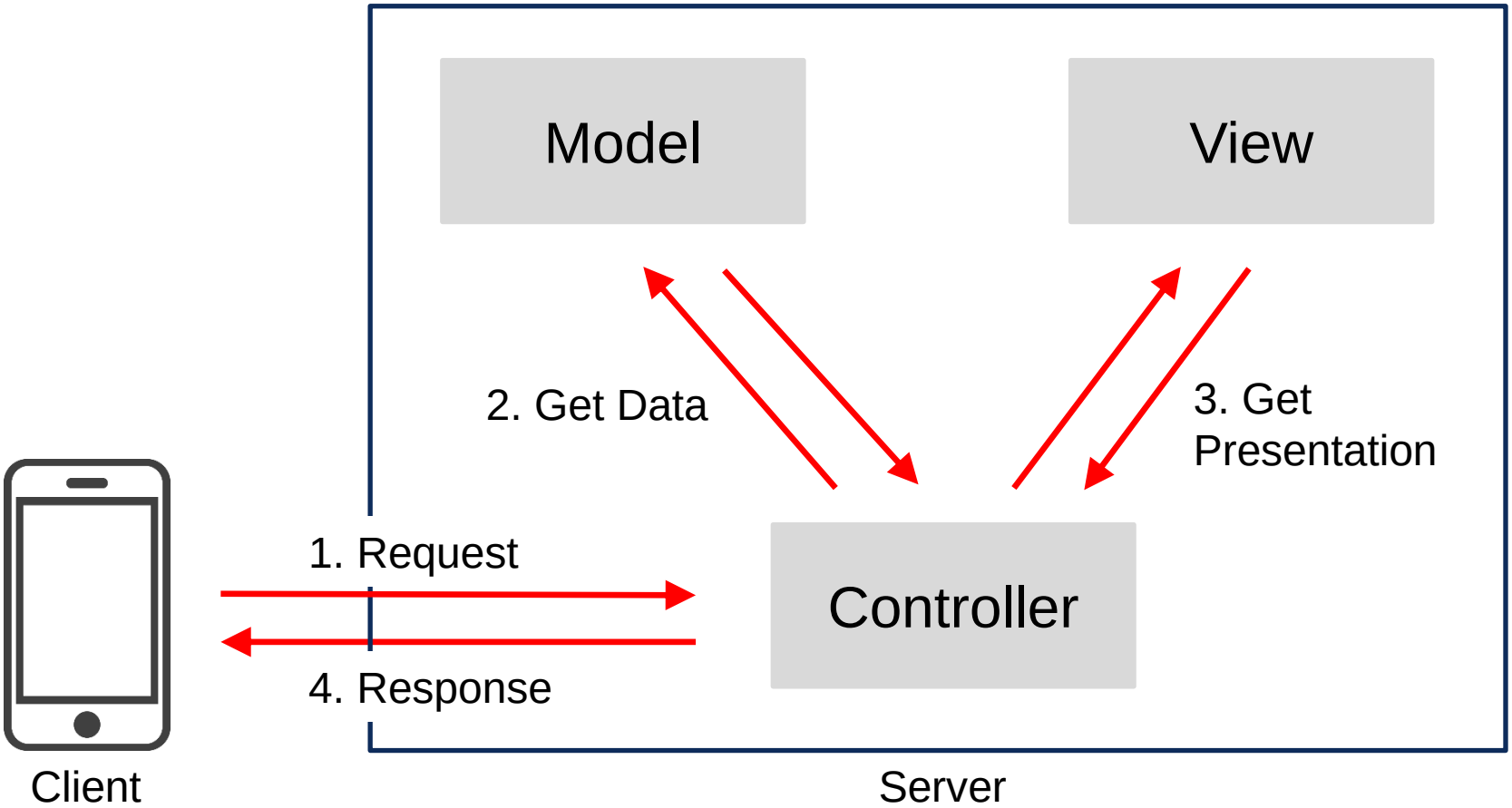
```
GET /api/TodoItems/10 HTTP/1.1
Host: localhost:59136
clientId: 77a06fd447b4
```

```
[Route("api/[controller]")]
public class TodoItemsController : ControllerBase {
  [HttpGet("{itemId}")]
  public TodoItem GetToDoItem(
      [FromHeader] string clientId, [FromRoute] long itemId) {
    Debug.WriteLine(clientId);
    Debug.WriteLine(itemId);
  }
}
```

[FromHeader]	A header value
[FromQuery]	a query string value
[FromRoute]	A query string value
[FromForm]	Form data posted in the body of the request

```
77a06fd447b4
10
```

Next Do you remember MVC?



Where is the **Binding Model** we're studying in this picture?

Topics

- HTTP Request revisit
- Model Binding
- **More details M in MVC (Self-Study)**
  - **Binding Model**
  - **Application Model**
  - **View Model**

Binding Model

Self study

What we study in this lecture is called **Binding Model**, which is the information **provided by the client** when it makes a request

```
public IActionResult Create(Movie movie)
{
   if (MovieService.IsValid(movie))
   {
      _movieDB.Add(movie);
      _movieDB.SaveChanges();
      return RedirectToAction(nameof(Index));
   }
   return View(movie);
}
```

As we've learned, Binding Models are passed to a controller's action method as parameters

Where do we expect data of ***movie*** object come from?

Application Model

Self study

This includes **anything needed** to perform the **business actions** in our apps

```
public IActionResult Create(Movie movie) {
   if (MovieService.IsValid(movie)) {
      _movieDB.Add(movie);
      _movieDB.SaveChanges();
      return RedirectToAction(nameof(Index));
   }
   return View(movie);
}
```

Application Model may include

- **Domain models** (represents the things our apps try to describe, e.g. the *Movie* class)
- **Database model** (represents the data stored in the database, e.g. *_movieDB* object and movie data in the DB)
- Other **additional services** (e.g. *MovieService* to validate the data)

Our business action is to create a new movie record. What are needed?

View Model

Self study

This contains the data **needed by the view** to generate the response

```
public IActionResult Details(int id)
{
   Movie movie = _movieDB.Movie.Find(id);
   List<Movie> relatedMovies =
                 MovieService.GetRelated(movie);

   ViewData["movie"] = movie;
   ViewData["relatedMovies"] = relatedMovies;
   return View();
}
```

For Web API, **View Model** is usually called **API Model** instead

In this example, what are needed for the view to generate the response?

Readings

- ASP.NET Core in Action, Chapter 6, The binding model: retrieving and validating user input (section 6.1, 6.2), *by Andrew Lock*
- Which properties to bind?
  - https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/implementing-basic-crud-functionality-with-the-entity-framework-in-asp-net-mvc-application#update-the-create-page
  - Preventing mass assignment or over posting https://andrewlock.net/preventing-mass-assignment-or-over-posting-with-razor-pages-in-asp-net-core/
- Model Binding in ASP.NET Core https://docs.microsoft.com/en-us/aspnet/core/mvc/models/model-binding?view=aspnetcore-6.0