ASP.NET Web API

Creating Web Services with C#

greenwich.edu.vn

UNIVERSITY of GREENWICH

Alliance with FPT Education

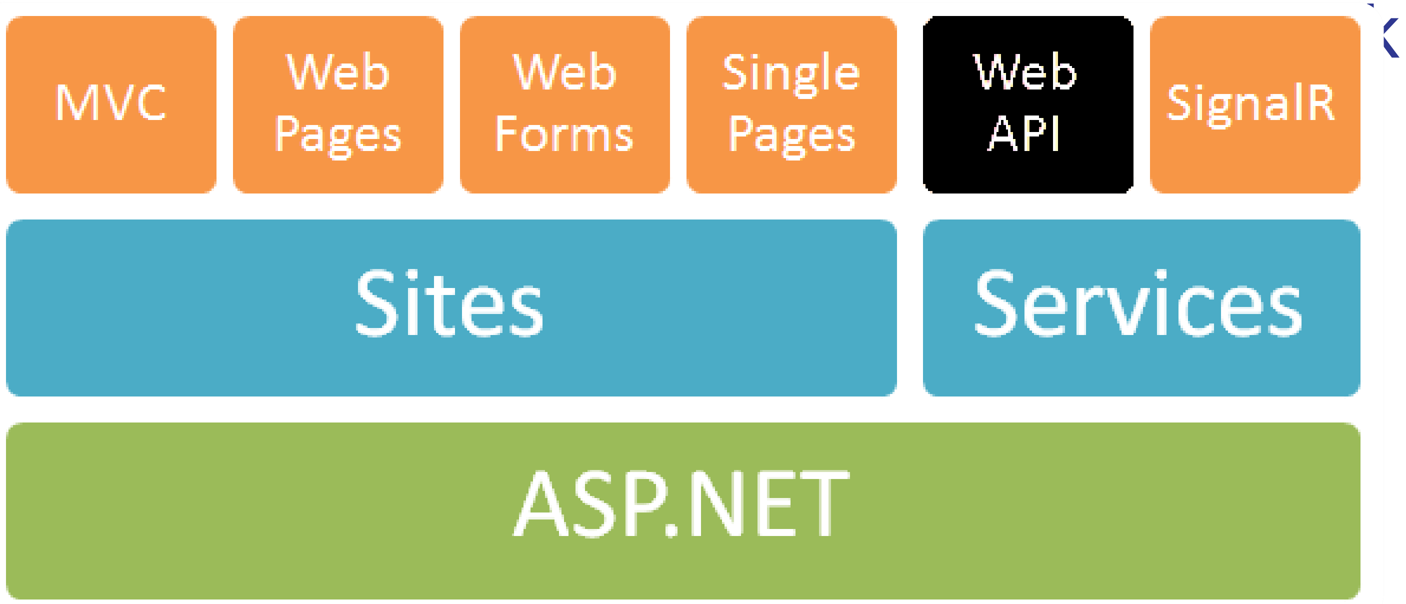
Table of Contents

1. What is ASP.NET Web API?
   – Web API Features
2. Web API Controllers
   – Actions
   – Routes
   – Return Types and Status Codes
   – Binding Models and View Models
   – Built-In User Identity

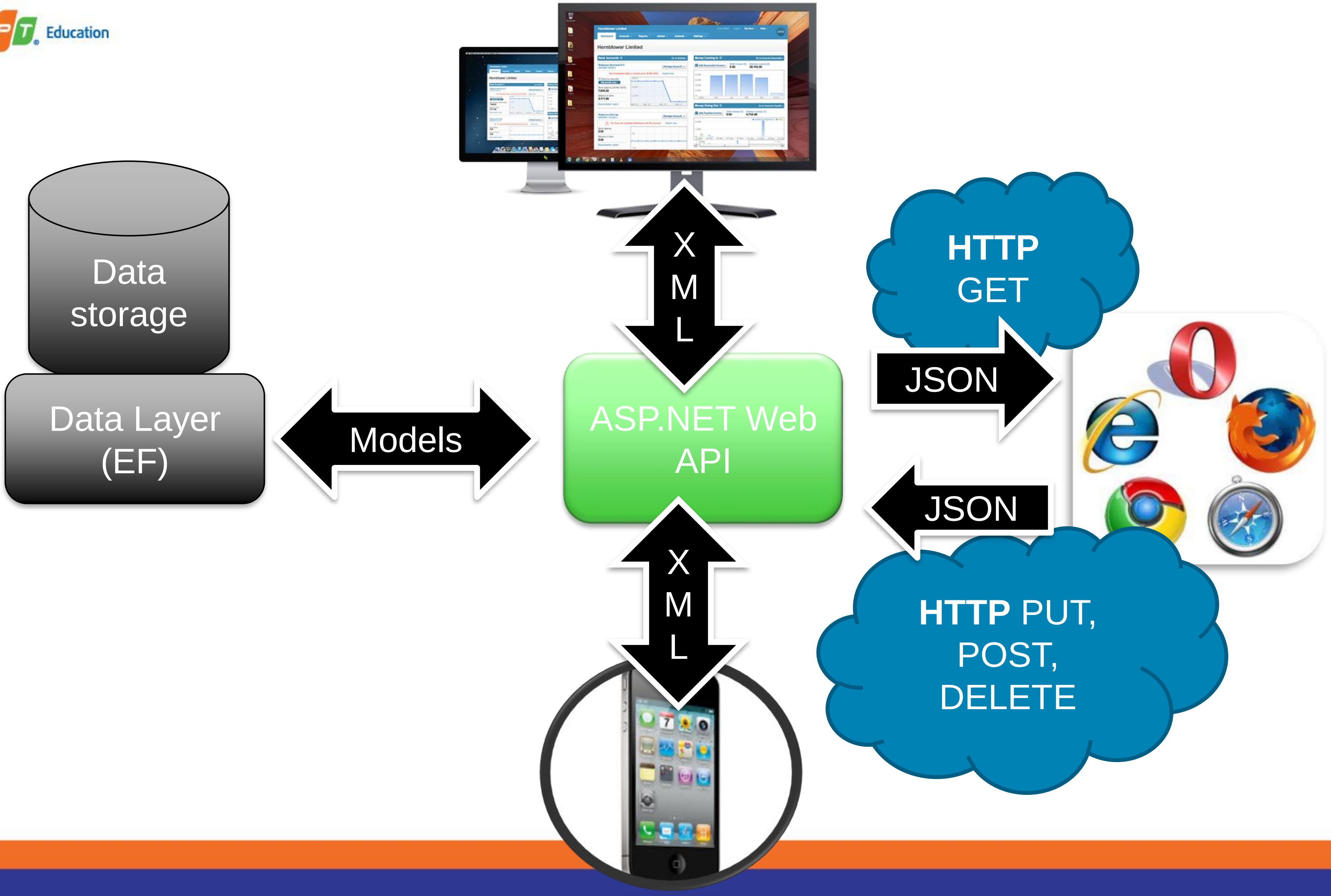
WHAT IS ASP.NET WEB API?

ASP.NET Web API

- ASP.NET Web API == platform for building **RESTful** Web services
  - Running over the **.NET** Framework



ASP.NET Web API



Web API Features

- Easy to use framework, very powerful
- Modern **HTTP** programming model
  - Access to strongly typed **HTTP** object model
  - **HttpClient API** – same programming model
- Content negotiation
  - Client and server negotiate about the right data format
  - Default support for **JSON**, **XML** and Form URL-encoded formats
  - We can add own formats and change content negotiation strategy

Web API Features (2)

- Query composition
  - Support automatic paging and sorting
  - Support querying via the **OData URL** conventions when we return **IQueryable<T>**
- Model binding and validation
  - Combine **HTTP** data in **POCO** models
  - Data validation via attributes
  - Supports the same model binding and validation infrastructure as **ASP.NET MVC**

Web API Features (3)

- **Routes** (mapping between **URIs** and code)
  - – Full set of routing capabilities supported within **ASP.NET (MVC)**
- **Filters**
  - – Easily decorates **Web API** with additional validation
    - Authorization, CORS, etc.
- **Testability**
- **IoC** and dependency injection support
- Flexible hosting (**IIS**, **Azure**, **self-hosting**)

ASP.NET Web API 2

- Attribute routing
- **OData** improvements: **`$select`**, **`$expand`**, **`$batch`**, **`$value`** and improved extensibility
- Request batching
- Portable ASP.NET Web API Client
- Improved testability
- **CORS** (Cross-origin resource sharing)
- Authentication filters
- **OWIN** support and integration (owin.org)

WEB API CONTROLLERS

Web API Controllers

- A **controller** class handles **HTTP** requests
  - Web API controllers derive from **`ApiController`**
  - **ASP.NET Web API** by default maps **HTTP** requests to specific methods called "actions"

Action	HTTP method	Relative URI	Method
Get a list of all posts	GET	/api/posts	Get()
Get a post by ID	GET	/api/posts/id	Get(int id)
Create a new post	POST	/api/posts	Post(PostModel value)
Update a post	PUT	/api/posts/id	Put(int id, PostModel value)
Delete a post	DELETE	/api/posts/id	Delete(int id)
Get a post by category	GET	/api/posts?category=news	Get(string category)

Actions

- Actions are public methods of a controller

```
public class PostsController : ApiController
{
    [HttpGet]
    public IEnumerable<string> GetPosts()
    {
        return new [] { "Hello", "WS&C deadline question.." };
    }

    [HttpPost]
    public void AddPost(string content)
    {
        // Add post to DB..
    }
}
```

Web API Request Processing

1. Web request is sent

```
http://localhost:1337/api/posts
```

2. Match controller from route

```
GET /api/posts HTTP/1.1
Host: localhost:1337
Cache-Control: no-cache
```

3. Controller Responds

```
HTTP/1.1 200 OK
Content-Length: 11
"some data"
```

```
public class PostsController : ApiController
{
    public string Get()
    {
        return "Some data";
    }

    public string Edit(Post post)
    {
        ...
    }
}

public class UsersController : ApiController
{
    ...
}
```

Routing

- Routing == matching **URI** to a controller + action
- Web API support the full set of routing capabilities from **ASP.NET (MVC)**
  - Route parameters
  - Constraints (using regular expressions)
  - Extensible with own conventions
  - Attribute routing is available in version 2

Attribute Routing

- Routing can be done through attributes
  - **`[RoutePrefix()]`** – annotates a controller route
  - **`[Route()]`** – annotates a route to an action
  - **`[HttpGet]`**,**`[HttpPut]`**, **`[HttpPost],`** etc. – specify the request method

```
[RoutePrefix("api/posts")]
public class PostsController : ApiController
{
  [Route("{id}")]
  public Post Get(int id)
  { ... }

  [HttpGet]
  [Route("{id}/likes")]
  public IEnumerable<Like> GetPostLikes(int id)
  { ... }
}
```

Default Route

- **Web API** also provides smart conventions by default
  - **HTTP Verb** is mapped to an action name
  - Configurations can be added in **`WebApiConfig.cs`**

```
config.Routes.MapHtpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RoutesParameter.Optional }
);
```

ETURN TYPES

T, IEnumerable<T>, IQuearyable<T>
IHttpActionResult

Return Types

- Actions can return several types
- Returned data automatically serialized to JSON or XML
  - **T** – generic type (can be anything)

```
public Comment GetCommentById(int id) { ... }
```

  - **IEnumerable<T>** - foreach-able collection of generic type T

```
public IEnumerable<Comment> GetPostComments(int id) { ... }
```

  - **IQueryable<T>** - collection of generic type T (supports filtering, sorting, paging, etc.)

```
public IQueryable<Comment> GetPostComments(int id) { ... }
```

Return Types (2)

- `void` – returns empty HTTP response 204 (No Content)
- `IHttpActionResult` – returns an abstract HTTP response with status code + data

```
public IHttpActionResult GetPostComments(int id)
{
    var context = new ForumContext();
    var post = context.Posts.FirstOrDefault(p => p.Id == id);
    if (post == null)
        return this.BadRequest("Invalid post id");

    return this.Ok(post);
}
```

200 OK + serialized data

HTTP Status Codes

- It's a good practice always to return a status code
  - **Return data with concrete status code method (e.g.** `Ok()`, `BadRequest()`, `NotFound()`, `Unauthorized()`, **etc.)**

```
var top10Users = context.Users.All()
     .Take(10)
     .Select(u => u.Username);

return this.Ok(top10Users);
```

  - **Return only status code**

```
return this.StatusCode(HttpStatusCode.Forbidden);
```

Model Binding

- By default the **Web API** will bind incoming data to **POCO (CLR)**
  - Will look in body, header and query string
  - Request data will be transferred to a C# object
    - E.g. the query string will be parsed to **`RegisterBindingModel`**

```
.../api/users/register?username=donjuan&password=12345
```

```
public IHttpActionResult Register(
    RegisterBindingModel user)
{
  string name = user.Username;
  ...
}
```

username	**donjuan**
password	**12345**

Binding Models

```
Request Method:POST
FormData content=Hello+Guys&
author=Gosho&categoryId=5
```

```
public class AddPostBindingModel
{
  public string Content { get; set; }
  public int AuthorId { get; set; }
  public int? Category { get; set; }
}
```

Validation attributes can be set in the binding model

```
[HttpPost]
public IHttpActionResult CreatePost(AddPostBindingModel postModel)
{
   if (!postModel.Category.HasValue) ...
}
```

Binding Model Validation

```
Request Method:POST
FormData username=Motikarq&password=#otpo6ti4kata
```

- `ModelState` holds information about the binding model

```
public class UsersController : ApiController
{
    public IHttpActionResult Register(RegisterBindingModel user)
    {
        if (!this.ModelState.IsValid)
            return this.BadRequest(this.ModelState);
        ...
    }
}
```

```
public class RegisterBindingModel
{
    [Required]
    public string Username { get; set; }
    [MinLength(6)]
    public string Password { get; set; }
    public int Age { get; set; }
}
```

Using Binding Models – Example

```
public IHttpActionResult AddPost(AddPostBindingModel postModel)
{
    if (!this.ModelState.IsValid)
        return this.BadRequest(this.ModelState);

    var context = new ForumContext();
    var author = context.Users
        .FirstOrDefault(u => u.Username == postModel.Author);
    if (author == null)
        return this.BadRequest("Author does not exist");

    context.Posts.Add(new Post() { Content = postModel.PostContent,
        Author = author, CategoryId = postModel.CategoryId });
    context.SaveChanges();

    return this.Ok(newPost);
}
```

Data Source Attributes

- Web API can specify request data source
  - `[FromUri]` – binds data from query string to action parameters

```
http://localhost:1337/api/posts/comments?page=5
```

```
public IHttpActionResult GetComments([FromUri]int page)
{...}
```

  - `[FromBody]` – binds data from request body to binding model

```
public IHttpActionResult Register(
    [FromBody]RegisterBindingModel user)
{ ... }
```

Media Type Formatters

- **MediaTypeFormatters** are used to bind both input and output
  - Mapped to content types
  - In **`WebApiConfig.cs`** we can configure the response to return JSON by default

```
config.Formatters.JsonFormatter.SupportedMediaTypes.Add(
    new MediaTypeHeaderValue("text/html"));
```

  - And JSON to follow camel case conventions

```
config.Formatters.JsonFormatter.SerializerSettings.ContractResolver =
    new CamelCasePropertyNamesContractResolver();
```

View Models

- View models are classes which represent data to be displayed
  - Used to project only needed data

```
[HttpGet]
public IHttpActionResult GetAllComments(int postId)
{
    var post = this.context.Posts.FirstOrdDefault(p => p.id == postId);
    if (post == null)
        return this.BadRequest("Invalid post id");

    var comments = post.Comments.Select(c => new CommentViewModel
        {
            Id = c.id
            Content = c.Content,
        });
    return this.Ok(comments);
}
```

```
public class CommentViewModel
{
    public int Id { get; set; }
    public string Content { get; set; }
}
```

ASP.NET IDENTITY API

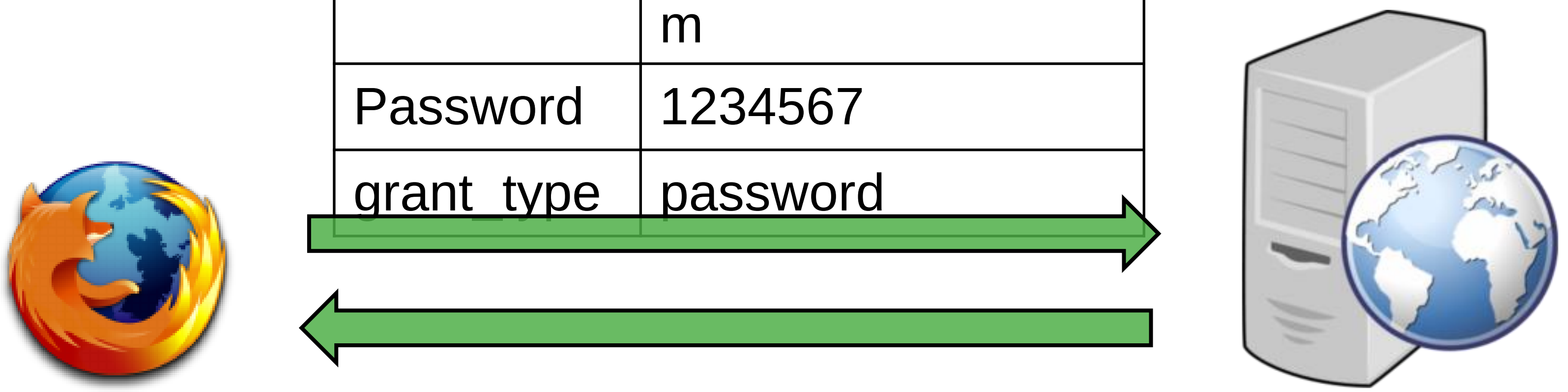
Setup, Registration, Login, Logout

ASP.NET Identity

- The ASP.NET Identity system
  - – Authentication and authorization system for ASP.NET Web apps
    - Supports ASP.NET MVC, Web API, Web Forms, SignalR, Web Pages
  - – Handles users, user profiles, login / logout, roles, etc.
  - – Based on the OWIN middleware (can run outside of IIS)
  - – Automatically integrated when the Individual User Accounts option is selected on Web API project creation

Identity Authentication (Login)

POST localhost:55602/Token	
Username	motikarq@gmail.co m
Password	1234567
grant_type	password

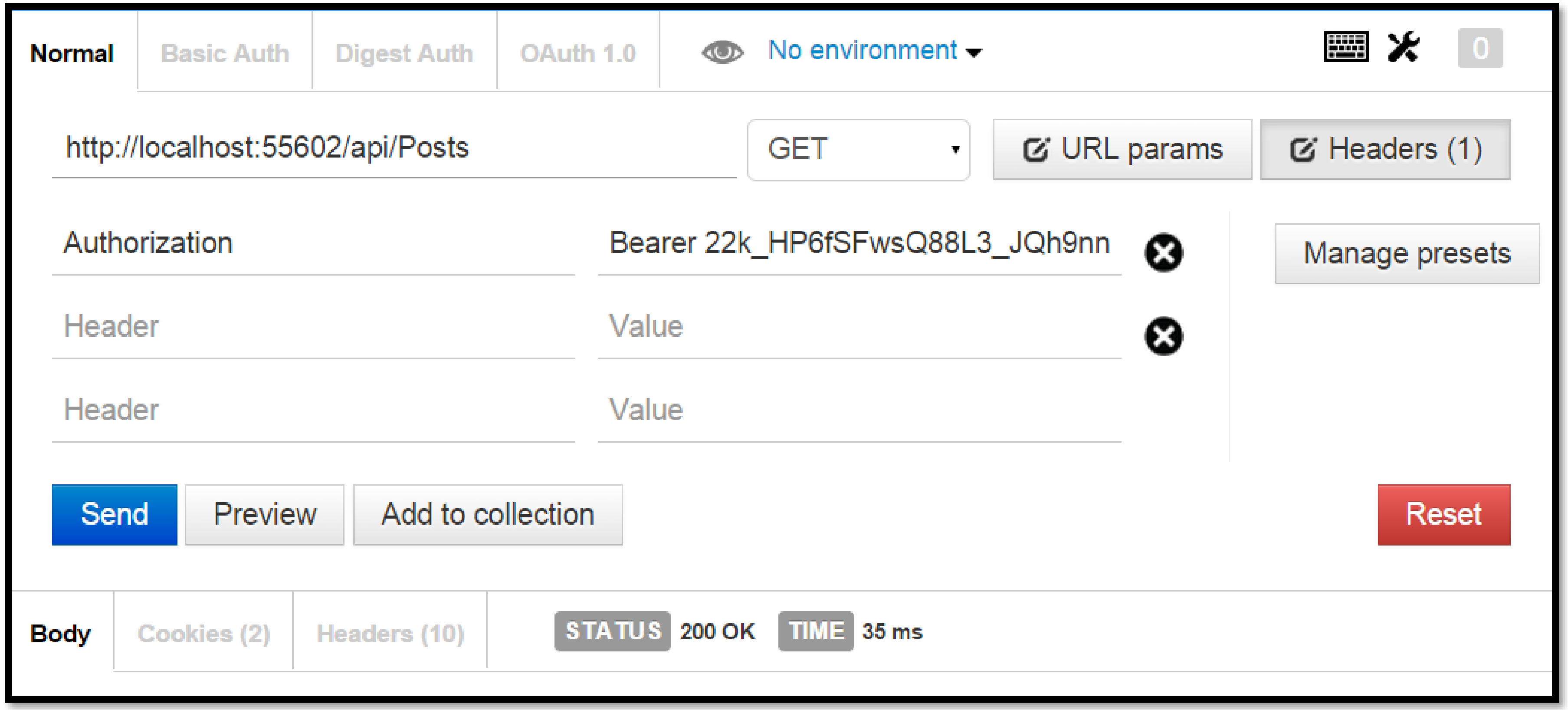


200 OK	
access_token	22k_HP6fSFwsQ88L3_JQh9nnx3…
token_type	bearer
expires_in	1209599
userName	jamal@hussein.com
.expires	Thu, 27 Aug 2015 12:42:38 GMT

Sent in future requests' headers for authentication

Request Authentication

- Access token should be put in request headers



http://localhost:55602/api/Posts

Authorization	Bearer 22k_HP6fSFwsQ88L3_JQh9nn
Header	Value
Header	Value

STATUS 200 OK TIME 35 ms

ASP.NET Authorization

- Use the `[Authorize]` and `[AllowAnonymous]` attributes to configure authorized / anonymous access for controller / action

```
[Authorize]
public class AccountController : ApiController
{
  // GET: /account/login (annonymous)
  [AllowAnonymous]
  public IHttpActionResult Login(LoginBindingModel model) { … }

  // POST: /account/logout (for logged-in users only)
  [HttpPost]
  public IHttpActionResult Logout() { … }
}
```

Check the Currently Logged-In User

```
// GET: /users/gosho (for logged-in users only)
[Authorize]
public IHttpActionResult GetUserInfo()
{
    string currentUserId = this.User.Identity.GetUserId();
    if (currentUserId == null)
    {
        return this.Unauthorized("Access denied");
    }
    ...
}
```