* Course Overview
  + Design your api with associations
* Introduction
  + Build api with asp.net web api
* How HTTP Works
  + Request: verb, headers, content
  + Response: status code, headers, content
  + Server is stateless
    - Connectionless
  + Verbs
    - Get: retrieve a resource
    - Post: add a new resource
    - Put: update an existing resource
    - Patch: update an existing resource with set of changes
    - Delete: removing the existing resource
  + REST
    - REpresentational State Transfer
      * Separation of client and server
      * Server Requests are Stateless
      * Cacheable Requests
      * Uniform Interface
    - Problems
      * Too difficult to be qualified as “REST”
      * Dogma of Rest vs Pragmatism
      * Structure architectural style
      * The need to be productive
* What are Resources
  + People, invoices, payments, products
  + Are resources just entities?
    - Resources has context
    - A person or an order in the order details
  + URIs
    - Uniform resource identifiers
    - Paths to resources
    - ex) ap.yourserver.com/people
  + Query strings for non-data elements
    - Format, sorting, searching
* Where We’re Headed
  + Camp -> location -> talks -> speaker
* Design the API
  + <http://.../api/camps>
  + <https://../api/camps/moniker>
  + <https://../api/camps/moniker/talks>
  + <https://../api/camps/moniker/talks>?topic=database
* Starting the Project
  + Get the starting code from ‘<https://github.com/psauthor/webapifundamentals>’
  + Global.asax
    - Set up that runs the configs files in App\_start
  + AutofacConfig
    - Dependency injection
* Using Postman
  + Download postman
  + Right click on project
    - Click properties
    - Click web
    - Click ‘Don’t open a page’
* Getting the Project Ready
  + In web.config
    - There is a connection string to the database
  + The database will be generated the first we need to access data
  + Click on view
    - Click on ‘SQL Server Object Explorer’
    - Check out the server
* What We’ve Learned
* Introduction
  + Request comes in with route in the form of a url
  + Search routes that match
  + Matches to action on a controller
  + Execution action
  + Return results as a response
* Creating an Action
  + Create a new class filer in the controller file
    - CampsController.cs
    - Have it inherit from ApiController
    - Ex) public class CampsController : ApiController
  + The routing will find a [name] + Controller and is an api controller
* Status Codes
  + Codes
    - 200: ok
    - 302: found
    - 400: bad request
    - 500: internal error
    - Etc
* Using Status Codes
  + To return payload and status codes return IHttpActionResult
  + ex)
    - public IHttpActionResult Get()
    - {
    - return Ok(new { Name = “Shawn”, Occupation = “Teacher });
    - }
* Using GET for Collections