Web API Design Guidelines

# Common Guidelines

* API should be having self-explementary and self-documentary
* Don’t include verbs in api method names. Like GetCustomers, GetOrders, DeteteOrder, etc.
* API should be having state less
* Use URI paths for resources and query string for non-data elements like, sorting, paging etc.

# URI Design

* Use Nouns not verbs
* Prefer Plurals for method names
  + api/Customers
  + api/Invoices
  + api/Orders
* Use Identifiers to locate individual items in Uris. This identifier does not have to be a primary key. It can be anything.
  + api/Customers/123 --> id
  + api/Invoices/2003-01-24 --> date

# How to use Verbs?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **resource** | **Get(read)** | **Post(Create)** | **Put(Update)** | **Delete(delete)** |
| **/customers** | Get List | Create Item | Update Batch | Error |
| **/customers/123** | Get Item | Error | Update Item | Delete Item |

# What should you return?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **resource** | **Get(read)** | **Post(Create)** | **Put(Update)** | **Delete(delete)** |
| **/customers** | List | New Item | Status Code Only | Status Code Only\* |
| **/customers/123** | Item | Status Code Only\* | Updated Item | Status Code Only |

\* = error status code

# Status Codes

Http defines status codes

|  |  |
| --- | --- |
| 200 - Ok | 400 - Bad Requst |
| 201 - Created | 401 - Not Authorized |
| 202 - Accepted | 403 - Forbidden |
| 302 - Found | 404 - Not Found |
| 304 - Not Modified | 405 - Method not allowed |
| 307 - Temp Redirect | 409 - Conflict |
| 308 - Perm Redirect | 500 - Internal Error |

At minimum API should use following Status codes

|  |  |
| --- | --- |
| 200 - Ok | It worked |
| 400 - Bad Request | You did bad |
| 500 - Internal error | We did bad |

API could also use following status codes

|  |  |
| --- | --- |
| 201 - Created |  |
| 304 - Not Modified |  |
| 404 - Not Found |  |
| 401 - Unauthorized |  |
| 403 - Forbidden |  |

# Associations

* For sub objects, use URI Navigations
  + api/Customers/123/Invoices
  + api/Invoices/2003-01-24/Payments
* API should return list of related objects or single object
* Result object shape of both of below api should have same schema.
  + api/Customers/123/Invoices
  + api/Invoices
* We May have multiple associations on same object
  + api/Customers/123/invoices
  + api/Customers/123/payments
  + api/Customers/123/shipments
* anything more complex should use query string. instead of creating simple entity for each scenario
  + api/customres?state=GA
  + api/customers?state=Ga&salesperson=144
  + api/customers/hasopenorders=true

# Designing Results

* Always return simple objects
* prefer camelCasing

# Designing Collections

Usually wrap collection into simple object. It will help to return other properties related to collection. E.g.

{

"count" : 345,

"result":[{}]

}

# Paging

* Lists should always support paging
* Use querystring parameters to request paging information
* Optional: Use object wrapper to indicate next/prev link

{

"totalResults":1598,

"nextPage":"api/customers/?page=3",

"prePage":"api/customers/?page=1",

"results":[]

}

* Make page size configurable
  + api/Customers?page=1&pageSize=10

# API Version

* We need a way to evolve the api without breaking existing clients. Versioning helps here.
* API versioning is not product versioning.
* Following techniques can be use for API versioning.
  + Content Negotiation (Popular Nowadays)
  + Custom Headers (Popular Nowadays)
  + URI Components (Very Common, easy to implement, but adds technical debt over the period)
  + Querystring parameters
* Versioning should have from the first release of the Api

# Security

* Secure In-Transit - SSL is appropriate
* Secure API itself - CORS Origin Security and Authorization/Authentication
* Use API Keys and Secrets for API clients.