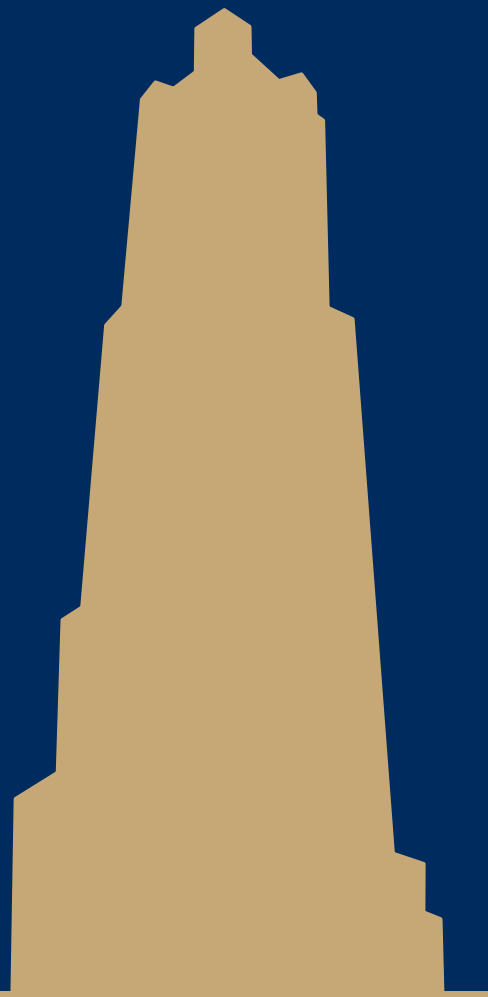
# CS/COE 1520

pitt.edu/~ach54/cs1520

REST



- When the web was designed the designers did a really good job

  - Roy Fielding

    (One of the designers of HTTP)

# What properties does the web exhibit?

- Resources exist and can be globally identified
  - E.g., URLs
- Representations of these resources are exchanged by clients and servers
  - E.g, HTML pages, JSON
- Standard interfaces are used for communication
  - E.g., HTTP
- It should not matter to clients if they are directly connected to a server or there are caches, NAT, other layers in between
- Client/server communications should be stateless

# Building large scale networked systems is hard

- So let's take those properties about the web and use that as the basis of an architectural style
- Representational state transfer
  - Systems that follow the same architectural principles are said to be RESTful
  - An application should be able to interact with a resource with only the following knowledge:
    - The identifier of the resource
    - The action to be performed
    - An understanding of the representation returned

# Interfaces should be simple

- GET
- POST
- PUT
- DELETE

-NOT-

- getUsers()
- getNewUsersSince(date SinceDate)
- savePurchaseOrder(string CustID, string PurchaseOrderID)

# Example:  Interface to a collection

- E.g., http://example.com/resources
- GET
  - List the URIs and perhaps other details of the collection's members
- PUT
  - Replace the entire collection with another collection
- POST
  - Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation
- DELETE
  - Delete the entire collection.

# Another Ex:  interface to an item in a collection

- E.g., http://example.com/resources/item37
- GET
  - Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type
- PUT
  - Replace the addressed member of the collection, or if it doesn't exist, create it
- POST
  - Not generally used
- DELETE
  - Delete the addressed member of the collection.

# OK, use HTTP and build RESTful systems, got it

POST /classScedService HTTP/1.1

<openClassRequest term="SUM2016" class="CS1520"/>

HTTP/1.1 200 OK

```
<openClassList>
        <class number="CS1520" time="930" />
        <class number="CS1520" time="1430" />
</openClassList>
```

POST /classScedService HTTP/1.1

```
<registerRequest>
        <class number="CS1520" time="1430" />
        <student id = "alice"/>
</registerRequest>
```

HTTP/1.1 200 OK

```
<registrationFailure>
<registration>
        <class number="CS1520" time="1430" />
        <student id="alice"/>
        <reason>Full Class</reason>
</registration>
</registrationFailure>
```

# Introducing more resources

POST /classes/SUM2016 HTTP/1.1

<openClassRequest class="CS1520"/>

HTTP/1.1 200 OK

```
<openClassList>
        <class id=123 number="CS1520" time="930" />
        <class id=124 number="CS1520" time="1430" />
</openClassList>
```

POST /registration/124 HTTP/1.1

```
<registerRequest>
        <student id = "alice"/>
</registerRequest>
```

HTTP/1.1 200 OK

```
<registrationFailure>
        <class id=124 number="CS1520" time="1430" />
        <student id="alice"/>
        <reason>Full Class</reason>
</registrationFailure>
```

# Using more HTTP verbs/response codes

GET /classes/SUM2016?status=open HTTP/1.1

HTTP/1.1 200 OK

```
<openClassList>
        <class id=123 number="CS1520" time="930" />
        <class id=124 number="CS1520" time="1430" />
</openClassList>
```

POST /registration/124 HTTP/1.1

```
<registerRequest>
        <student id = "alice"/>
</registerRequest>
```

HTTP/1.1 201 CREATED Location=/registrations/124/registration44

HTTP/1.1 409 CONFLICT

```
<registration>
        <openClassList>
                <class id=123 number="CS1520" time="930" />
                <class id=124 number="CS1520" time="1430" />
        </openClassList>
        <student id = "alice"/>
</registration>
```

# Alright, NOW we're RESTful, right?

- How does the client know that /registration/124 is the URI for registering for class id 124?
  - This wasn't mentioned in the openClassList response
  - Violates our assumption of what the client should need to know to interact with our system!

# HATEOAS

- Hypermedia as the Engine of Application State
  - Clients should interact with a network application entirely through hypermedia provided dynamically by application servers
    - No prior knowledge of application interaction required

# HATEOAS in action

GET /classes/SUM2016?status=open HTTP/1.1

HTTP/1.1 200 OK

```
<openClassList>
        <class id=123 number="CS1520" time="930">
                <link rel = "/linkrels/class/register"
                        uri = "/registrations/123"/>
        </class>
        <class id=124 number="CS1520" time="1430">
                <link rel = "/linkrels/class/register"
                        uri = "/registrations/124"/>
        </class>
</openClassList>
```

# HATEOAS in action 2

POST /registration/124 HTTP/1.1

```
<registerRequest>
        <student id = "alice"/>
</registerRequest>
```

HTTP/1.1 201 CREATED Location=/registrations/124/registration44

```
<registration>
        <class id=124 number="CS1520" time="1430" />
        <student id = "alice"/>
        <link rel = "/linkrels/registration/drop"
                uri = "/registrations/124/registration44"/>
        <link rel = "/linkrels/registration/setgradeoption"
                uri = "/registrations/124/registration44/gradeOpt"/>
</registration>
```

# Why HATEOAS?

- Assume clients enter your app through a simple fixed URL
  - Alice could have just gone to registrar.pitt.edu
  - All future actions the client may take are discovered within resource representations returned from the server
    - All that's needed is basic knowledge of hypermedia
    - Alice could be given the link to a list of open classes from the registrar index resource, leading to the previous examples
    - Further, the registrar could update how the registration application works, but Alice wouldn't have to update her client to respect these changes

# Something something theory and practice

- Applications that call themselves RESTful may actually fall somewhere in between these 4 examples
  - May even basically embody the first example...