

# In this lecture, we will discuss...

## ✧ HTTP Methods

- POST
- PUT
- PATCH
- HEAD

# HTTP Methods - POST

- ✧ POST is for creating new resource instances
  - POST to a resource URI
  - Provide JSON payload (but optional)
  - Provide MIME type of the payload in the Content-Type header

# HTTP Methods - POST

```
> MoviesWS.post("/movies.json", :body=>{:movie=>{:id=>"123457", :title=>"rocky27"}}.to_json)
```

```
<- "POST /movies.json HTTP/1.1\r\n  
Content-Type: application/json\r\n  
Connection: close\r\n  
Host: localhost:3000\r\n  
Content-Length: 43\r\n  
\r\n"  
<- "{\"movie\":{\"id\":\"123457\",\"title\":\"rocky27\"}}"
```



# POST (Update) - Action

- ✧ Builds a **white-list** version of parameter hash
- ✧ Builds a new instance of the Movie class with the hash passed
- ✧ Saves the resultant Movie to the database
- ✧ Renders a result back to the caller based on the format requested in the response and the status of the save.



# PUT

- ✧ PUT is for **replacing** the data (Update)
- ✧ The Client
  - issues a PUT request
  - issues the request to `/movies/123457` URI
  - provides a JSON payload for update
  - provides `application/json` MIME type

# HTTP Methods - PUT

```
> response=MoviesWS.put("/movies/123457.json",:body=>{:movie=>{:title=>"rocky2700",:foo=>"bar"}}).to_json)
```

```
<- "PUT /movies/123457.json HTTP/1.1\r\n  
Content-Type: application/json\r\n  
Connection: close\r\n  
Host: localhost:3000\r\n  
Content-Length: 43\r\n  
\r\n"  
<- "{\"movie\":{\"title\":\"rocky2700\",\"foo\":\"bar\"}}"
```



# PUT(Update) - Action

- ✧ PUT expects the primary key to be in the `:id` parameter
- ✧ If the movie is found, processing continues
- ✧ Builds a **white-list-checked** set of parameters
- ✧ Supplies the values to the update method
- ✧ Returns the result document

# HTTP Methods - PATCH

- ✧ PATCH is for **partially** updating a resource
- ✧ Update a field vs. entire resource

```
MoviesWS.patch("/movies/123457.json", :body=>{:movie=>{:title=>"rocky2702", :foo=>"bar"}}.to_json)
```





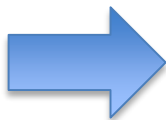
# HTTP Methods - HEAD

- ✧ HEAD is basically GET without the response body
- ✧ Useful to retrieve meta-information written in response headers
- ✧ Issue GET and store `Etag` for comparison later



# HEAD

```
> response=MoviesWS.get("/movies/123457.json")  
> response.header["etag"]  
=> "W/\"4cff78bec23ff12c4af51a97719009f1\""  
> doc=response.parsed_response
```



```
> response=MoviesWS.head("/movies/123457.json")  
> response.header["etag"]  
=> "W/\"4cff78bec23ff12c4af51a97719009f1\""  
> doc=response.parsed_response  
=> nil
```

# HTTP Methods - DELETE

- ✧ DELETE is for **deleting** a resource
- ✧ It accepts an `:id` parameter from the URI and removes that document from the database.
- ✧ No request body

# DELETE - Example

```
> response=MoviesWS.delete("/movies/123457.json")
> response.response
=> #<Net::HTTPNoContent 204 No Content readbody=true>
> response.response.code
=> "204"
> doc=response.parsed_response
=> nil
```

# HTTP Methods - GET

- ✧ GET is for data **retrieval** only
- ✧ Free of side effects, a property also known as *idempotence* (discussed later)

```
> MoviesWS.get("/movies.json?title=rocky25&foo=1&bar=2&baz=3").parsed_response  
=> [{"id"=>"12345", "title"=>"rocky25", "url"=>"http://localhost:3000/movies/12345.json"}]
```



# Summary

- ✧ HTTP Methods maps seamlessly to CRUD operations
- ✧ Elegant and easy for the clients

## What's Next?

- ✧ Idempotence



# Next Topic.....

## Idempotence

