Outline

- What is REST ?
- HTTP and REST
- **RestFul APIs**

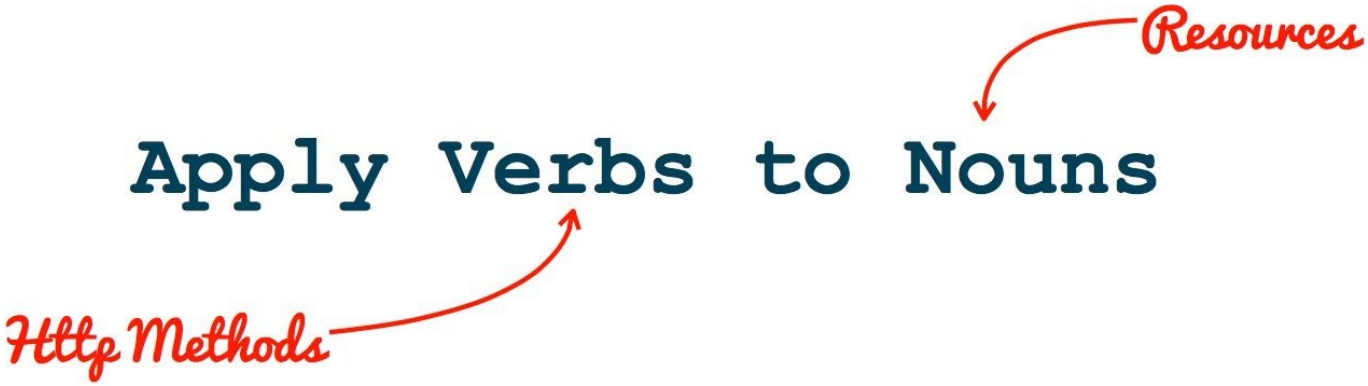
Web API

- Application program interface (API) to a defined request-response message system between clients and servers
  - Accessible via standard HTTP methods
- Request URLs that transfer representations (JSON, XML)

Restful APIs: Features

- Application program interface to a defined request-response message system between clients and servers
- Accessible via standard HTTP methods
- Request URLs that transfer representations (JSON, XML)

Designing Restful APIs



Collections

<VERB> http://example.com/users

GET Return all the objects in the collection

POST Create a new entry in the collection; automatically assign new URI and return it

PUT and **DELETE** not generally used

Elements

```
VERB> http://example.com/users/123
```

GET Return the specific object in collection

PUT Replace object with another one

DELETE Delete element

Using Parameters for Queries

```
<VERB> http://example.com/users/12345?
  where={"num_posts":{"$gt":100}}}
```

other parameters can be used to select fields, sort, etc.

parameters can also be URL-encoded

CheckList: Restful APIs

- Use nouns (but no verbs) as resources in URLs.
- Only expose relevant nouns
- GET method and query parameters should not alter the state (safe)
- PUT and DELETE methods should be idempotent (be applied only once on the server)

Class Activity

- Design a simple REST API to perform the following actions in a **Phonebook** application
  - Retrieve the list of all contacts in the phonebook
  - Retrieve a specific contact given their key
  - Retrieve the info of a specific contact given their first name and last name
  - Add a new contact to the phonebook
  - Modify the details of an existing contact
  - Remove a contact from the phonebook

Solution to the Activity - Retrieval

Use **nouns** rather than verbs

- To request all contacts, use
  - GET foo.com/contacts
- To request a specific contact given a key, use
  - GET foo.com/contacts/12345
- To find a contact (by first-name and last name),
  - GET foo.com/contacts?fname="ABC"&lname="XYZ"

Solution to the Activity – Add

Add a new contact to the phonebook

Add should be a **POST** request as it modifies the state of contacts, and is not idempotent

POST foo.com/contacts/

Send contact details in the body of the request, as JSON formatted object (say)

NOTE: We Post on the collection contacts

Solution to the Activity - Modify

Can use PUT if key is known (better than POST as it's idempotent). Can also use PATCH for partial updates to save bandwidth, if needed.

PUT foo.com/contacts/12345

Send the new data (to be modified) in the body of the PUT request – assumes key is present

Solution to the Activity – Delete

Use Delete method in HTTP to remove the object given its key (idempotent). Should not do anything if contact is not present in server.

DELETE foo.com/contacts/12345

can also be used for multiple contacts as follows

DELETE foo.com/contacts?firstName=“Jack”

OPEN API

REST API description language formerly known as "Swagger".

- Describe RESTful HTTP APIs in a machine-readable way
- Formally define a schema with endpoints of REST APIs and responses
- Communication vehicle between service providers and clients

```yaml
openapi: "3.0.0"
info:
  version: 1.0.0
  title: Petstore
  license:
    name: MIT
servers:
  - url: http://petstore.swagger.io/v1
paths:
  /pets:
    get:
      summary: List all pets
      operationId: listPets
      tags:
        - pets
      parameters:
        - name: limit
          in: query
          description: How many items to return at one time (max 100)
          required: false
          schema:
            type: integer
            format: int32
```

```
responses:
        200:
          description: A paged array of pets
          headers:
            x-next:
              description: A link to the next page of responses
              schema:
                type: string
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Pets"
```

```yaml
components:
  schemas:
    Pet:
      required:
        - id
        - name
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
    Pets:
      type: array
      items:
        $ref: "#/components/schemas/Pet"
```

```
components:
  schemas:
    Pet:
      required:
        - id
        - name
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
        tag:
          type: string
    Pets:
      type: array
      items:
        $ref: "#/components/schemas/Pet"
```

Resources

- https://www.openapis.org
- https://apievangelist.com
- https://speccy.io
- https://github.com/Rebilly/ReDoc
- https://openapi.tools
- https://github.com/openapitools/openapi-generator

Controller's role in Model, View, Controller

- Controller's job to fetch model for the view
  - May have other server communication needs as well (e.g. authentication services)
- Browser is already talking to a web server, ask it for the model
- Early approach: have the browser do a HTTP request for the model
  - First people at Microsoft liked XML so the DOM extension got called: **`XMLHttpRequest`**
- Allowed JavaScript to do a HTTP request without inserting DOM elements
- Widely used and called **AJAX** - **A**synchronous **J**avaScript **a**nd **X**ML
- Since it is using an HTTP request it can carry XML or anything else
  - More often used with JSON

XMLHttpRequest

Sending a Request

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange = xhrHandler;
xhr.open("GET", url);
xhr.send();
```

Any HTTP method (GET, POST, etc.) possible.

Responses/errors come in as events

XMLHttpRequest: status codes?

Sending a Request

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange = xhrHandler;
xhr.open("GET", url);
xhr.send();
```

Any HTTP method (GET, POST, etc.) possible.

Responses/errors come in as events

XMLHttpRequest: status codes?

Sending a Request

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange = xhrHandler;
xhr.open("GET", url);
xhr.send();
```

Any HTTP method (GET, POST, etc.) possible.

Responses/errors come in as events

200 OK
request succeeded, requested object later in this message

301 Moved Permanently
requested object moved, new location specified later in this message (Location:)

400 Bad Request
request message not understood by server

404 Not Found
requested document not found on this server

505 HTTP Version Not Supported

XMLHttpRequest

Sending a Request

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange = xhrHandler;
xhr.open("GET", url);
xhr.send();
```

Any HTTP method (GET, POST, etc.) possible.

Responses/errors come in as events

Event handling

```
function xhrHandler(event) {
  // this === xhr
  if (this.readyState != 4) { // DONE
      return;
  }
  if (this.status != 200) { // OK
      return; // Handle error ...
  }
  ...
  let text = this.responseText;
  ...
```

XMLHttpRequest event processing

- Event handler gets called at various stages in the processing of the request

0	UNSENT	open() has not been called yet.
1	OPENED	send() has been called.
2	HEADERS_RECEIVED	send() has been called, and headers and status are available.
3	LOADING	Downloading; responseText holds partial data.
4	**DONE**	**The operation is complete**.

- Response available as:

  raw text - `responseText`

  XML document - `reponseXML`

- Can set request headers and read response headers

XMLHttpRequest

Sending a Request

```
xhr = new XMLHttpRequest();
xhr.onreadystatechange = xhrHandler;
xhr.open("GET", url);
xhr.send();
```

Any HTTP method (GET, POST, etc.) possible.

Responses/errors come in as events

Event handling

```
function xhrHandler(event) {
  // this === xhr
  if (this.readyState != 4) { // DONE?
      return;
  }
  if (this.status != 200) { // OK?
      return; // Handle error ...
  }
  ...
  let text = this.responseText;
  ...
```

Traditional AJAX uses patterns

- Response is HTML

  ```
  elem.innerHTML = xhr.responseText;
  ```

- Response is JavaScript

  ```
  eval(xhr.responseText);
  ```

Neither of the above are the modern JavaScript framework way:

- Response is model data (JSON frequently uses here)

  ```
  JSON.parse(xhr.responseText);
  ```

Fetching resources with XMLHttpRequest via REST

- Can encode model selection information in request in:

URL path:

```
xhr.open("GET","userModel/78237489/fullname");
```

Query params:

```
xhr.open("GET","userModel?id=78237489&type=fullname");
```

Request body:

```
xhr.open("POST", url);
xhr.setRequestHeader("Content-type",
        "application/x-www-form-urlencoded");
xhr.send("id=78237489&type=fullname");
```

Other Transports: HTML5 WebSockets

- Rather than running over HTTP, HTML5 brings sockets to the browser
  - TCP connection from JavaScript to backend Web Server - Bidirectional pipes
- Event-based interface like XMLHttpRequest:

```
let socket = new WebSocket("ws://www.example.com/socketserver");
socket.onopen = function (event) {
  socket.send(JSON.stringify(request));
};

socket.onmessage = function (event) {
  JSON.parse(event.data);
};
```

Trending approach: GraphQL

- Standard protocol for backends from Facebook
  - Like REST, server exports resources that can be fetched by the web app
  - Unlike REST
    - GraphQL is a **query language** for APIs and a **runtime** for executing those queries by using a type system you define for the data.
    - Exports a "schema" describing the resources and supported queries.
    - Client specifies what properties of the resource it is interested in retrieving.
    - Unlike REST, which uses multiple endpoints to retrieve different data, GraphQL typically exposes a **single** endpoint.
- Gaining in popularity particularly compared to REST
  - Gives a program accessible backend - **Application Programming Interface (API)**

Questions?