Ajax Solution, Java Script, XHTMLHttpRequest and Response, HTML, CSS, JSON, iFrames, Settings of Java Script in Django, jQuery and Basic AJAX, jQuery AJAX Facilities, Using jQuery UI Autocomplete in Django

1. Explain XHTML Http Request and Response.
2. Develop a registration page for student enrolment without page refresh using AJAX.
3. Discuss how the setting of Javascript in Django.
4. Develop a search application in Django using AJAX that displays courses enrolled by a student being searched.

Ajax Solution

- **AJAX** stands for **Asynchronous JavaScript and XML**.
- AJAX is a technique used in web development to create asynchronous web applications. This allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes.
- This means that it is possible to update parts of a web page, without reloading the whole page.

How AJAX Works:

1. **Client-Side Event:**
   - A client-side event occurs that triggers an AJAX call. This could be a user action like clicking a button or loading a webpage.
2. **Create an XMLHttpRequest Object:**
   - The client-side JavaScript creates an XMLHttpRequest object to interact with the server.
3. **Configure the Request:**
   - The XMLHttpRequest object is configured with the type of request (GET or POST), the URL to send the request to, and whether the request should be asynchronous.
4. **Send the Request:**
   - The request is sent to the server.
5. **Server Processes the Request:**
   - The server processes the request and sends back the appropriate response, which could be data in XML, JSON, HTML, or plain text.
6. **Receive the Response:**
   - The client-side JavaScript receives the response from the server.
7. **Update the Web Page:**
   - The client-side JavaScript updates the web page content based on the response data without reloading the entire page.

Human Speech: An Overlaid Function:

- **Analogy Explanation:**
    - Human speech involves multiple body parts (tongue, lungs, diaphragm, brain) that perform primary functions (tasting, breathing, thinking) but also support the overlaid function of speech.
    - Similarly, Ajax overlays on existing web technologies (HTML, JavaScript, CSS) to add new functionality without replacing their primary purposes.

Ajax: Another Overlaid Function:

- **Core Functionality:**
    - Ajax is used to communicate with a server asynchronously, allowing for partial page updates.
    - It contrasts with the older paradigm where a full page reload was required for any update.

Underlying Technologies of AJAX

1. JavaScript:

- **Role in AJAX:** JavaScript is the core programming language used to create and control the XMLHttpRequest object, which is essential for AJAX operations. It handles the client-side interactions, sends the request to the server, and processes the server's response to update the web page dynamically.
- **Role:**
    - Central to Ajax, it interacts with the server using the XMLHttpRequest object.
    - Manages dynamic content updates on the web page.
- **Diadvantages:**
    - Language quirks (e.g., variable scope issues).
    - Inconsistent browser implementations necessitate cross-browser testing.
- **Advantages:**
    - Multiparadigm language supporting object-oriented and functional programming.
    - Dynamic and flexible, similar to Python in allowing on-the-fly changes.

Example:

```
var xhr = new XMLHttpRequest();

xhr.open('GET', 'https://api.example.com/data', true);

xhr.onreadystatechange = function() {
    if (xhr.readyState === 4 && xhr.status === 200) {
        document.getElementById('content').innerHTML = xhr.responseText;
    }
};
xhr.send();
```

2. XMLHttpRequest Object:

- **Definition:** The XMLHttpRequest object is a built-in browser object that allows JavaScript to send HTTP requests to a server and receive responses from it.
- **Properties:**
    - onreadystatechange: Event handler for changes in the request's state.
    - readyState: Represents the current state of the request.
    - responseText: Contains the response as a text string.
    - responseXML: Contains the response as an XML document.
    - status: HTTP status code of the response.
    - statusText: HTTP status text of the response.

    - xhr.readyState: Represents the state of the request (0 to 4).
        - 0: Uninitialized
        - 1: Open
        - 2: Sent
        - 3: In Progress
        - 4: Complete
    - xhr.status: HTTP status code returned by the server (e.g., 200 for success).
    - xhr.responseText: The response data as a string.

3. HTML/XHTML:

- **Role:**
    - Core markup languages of the web.
    - Provide the structure and content for web pages.
- **Integration:**
    - Ajax uses JavaScript to dynamically update parts of an HTML/XHTML document

4. CSS:

- **Role:**
  - Styles the web page.
- **Integration:**
  - JavaScript can manipulate CSS to reflect changes without reloading the page.

5. JSON and XML:

- **Data Formats:**
  - JSON (JavaScript Object Notation): Lightweight data-interchange format, often used in place of XML due to its simplicity and efficiency.
  - XML (eXtensible Markup Language): Flexible data format, initially the primary data format for Ajax.

6. DOM (Document Object Model):

- **Definition:** The DOM is a programming interface for web documents. It represents the structure of a document as a tree of objects that can be manipulated with JavaScript.
- **Role in AJAX:** The DOM allows JavaScript to dynamically update the content of a web page based on the server's response.

AJAX Variations Iframes

1. Comet

- **Definition:** Comet is an advanced technique in AJAX where the connection between the client and the server is kept open, allowing the server to push data to the client in real-time.
- **Usage:** Commonly used in applications like instant messaging where real-time updates are crucial.
- **How It Works:**
  - The connection (usually XMLHttpRequest) stays open indefinitely.
  - Alternatively, a new connection is opened whenever the old one is closed.
  - This allows the server to send new data to the client without the client having to request it.

2. Iframes

What is an Iframe?

- **Definition:** An iframe (short for inline frame) is an HTML element that allows you to embed another HTML document within the current webpage.
- **Purpose:** It lets you load and display content from another source (like a separate web page) without leaving the current page.

How Iframes Work

- **Structure:** An iframe is like a small window embedded in your web page that can show a completely different webpage.
- **Usage:** You can think of it as a picture-in-picture feature on your TV, where you can watch a small screen inside the main screen.

Why Use Iframes?

- **Partial Page Updates:** Iframes allow parts of a webpage to be updated without refreshing the whole page. This can make web applications faster and more responsive.
- **Example:** In Gmail, when you click to open an email or compose a new message, the content loads inside an iframe. This way, only the necessary part of the page updates, and you don't lose your place or history.

Benefits of Iframes

- **Browser History:** Actions within iframes can create entries in the browser's history. This means that if you navigate inside an iframe and then click the browser's back button, it will only affect the iframe's content rather than the whole page.
- **Seamless Integration:** Iframes can make changes and updates look smooth and seamless, providing a better user experience.

JavaScript/AJAX Libraries

1. jQuery

- **Why Use It:** jQuery simplifies JavaScript programming, reduces boilerplate code, and provides a consistent interface.
- **Benefits:**
  - Makes it easier to handle HTML document manipulation, event handling, and AJAX interactions.
  - Provides ready-made widgets (like sliders) for easy integration.
  - Easier to learn and use compared to other libraries.

Setting Up Javascript in Django

1. **Create the Static Directory**

   - Create a directory named `static` within your Django project directory.
   - Inside the `static` directory, create subdirectories for CSS, JavaScript, and images

```
mkdir -p static/css static/js static/images
```

2. **Edit `settings.py`**

   - Open your `settings.py` file and add the following configurations.

```
import os
# Get the absolute path of the directory containing the settings file
DIRNAME = os.path.abspath(os.path.dirname(__file__))
# Define the root directory for static files
MEDIA_ROOT = os.path.join(DIRNAME, 'static/')
# Define the URL path to access static files
MEDIA_URL = '/static/'
```

3. **Configure Static Files Serving in Development**

   - Still in `settings.py`, add a condition at the end to serve static files when `DEBUG` is `True`

```
from django.conf import settings
from django.conf.urls.static import static

if settings.DEBUG:

    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

jQuery and Basic AJAX

Create and Send an Ajax Request: Here, an Ajax request is sent to the server, and the response is displayed in a paragraph with the ID result.

```
var xhr = new XMLHttpRequest();

xhr.open("GET", "/project/server.cgi?text=world");

xhr.onreadystatechange = function() {

   if (xhr.readyState === 4 && xhr.status >= 200 && xhr.status < 300) {

       document.getElementById("result").innerHTML = xhr.responseText;

   }

};

xhr.send(null);
```

jQuery Ajax Example

jQuery simplifies the process with fewer lines of code and easier syntax.

1. **Include jQuery:** First, include jQuery in your HTML file.

   ```
   <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
   ```

2. **Simple Ajax Request Using jQuery:** The following line achieves the same functionality as the plain JavaScript example.

   ```
   $("#result").load("/project/server.cgi", "text=world");
   ```

Understanding jQuery Wrapped Sets

jQuery operations are performed on "wrapped sets," which are essentially collections of DOM elements with associated methods for manipulation.

Examples of Creating Wrapped Sets

1. **By ID:**

   ```
   $("#result"); // Selects the element with ID 'result'
   ```

2. **By Tag:**

   ```
   $("p"); // Selects all paragraph elements
   ```

3. **By Class:**

   ```
   $("p.summary"); // Selects all paragraph elements with class 'summary'
   ```

4. **By Descendant:**

   ```
   $("p a"); // Selects all anchor tags within paragraph elements
   ```

5. **By Direct Child**

   ```
   $("p > a"); // Selects all anchor tags whose immediate parent is a paragraph element
   ```

Chaining Operations:

jQuery allows chaining operations on wrapped sets, making the code concise and readable.

1. **Add Class and Hide Elements:**

   ```
   $("table.striped tr:even").addClass("even").hide("slow").delay(1000).show("slow");
   ```

Explanation:

- Select even-numbered rows from tables with the class `striped`.
- Add the class `even` to these rows.
- Slowly hide these rows.
- Wait for 1000 milliseconds.
- Slowly show these rows again.

jQuery AJAX Facilities

jQuery provides powerful facilities for AJAX operations, simplifying many common tasks. The core function for AJAX operations in jQuery is $.ajax(), with several convenience methods like $.get(), $.post(), and .load(). Here's an in-depth look at these functions and their usage.

$.ajax()

`$.ajax()` is the foundational method for making AJAX requests in jQuery. It offers the most control and flexibility by accepting a wide range of options.

```
$.ajax({
   data: "surname=Smith&cartTotal=12.34",
   dataType: "text",
   error: function(XMLHttpRequest, textStatus, errorThrown) {
      displayErrorMessage("An error has occurred: " + textStatus);
   },
   success: function(data, textStatus, XMLHttpRequest) {
      try {
         updatePage(JSON.parse(data));
      } catch(error) {
         displayErrorMessage("There was an error updating your shopping cart. Please call customer
service at 800-555-1212.");
      }
   },
   type: "POST",
   url: "/update-user"
});
```

Key Options:

- **data:** The data to be sent to the server, either as a query string or an object.
- **dataType:** The expected data type of the response (e.g., "text", "json", "html").
- **error:** A callback function that is executed if the request fails.
- **success:** A callback function that is executed if the request succeeds.
- **type:** The HTTP method to use for the request (e.g., "GET", "POST").
- **url:** The URL to send the request to.

Context: The `context` option allows you to set the `this` value in your callback functions, which can be useful for maintaining access to certain variables.

Example:

```
$.ajax({
   success: function(data, textStatus, XMLHttpRequest) {
      alert(this.name + ", your email address is " + this.email + ".");
      processData(data, this.name, this.email);
   },
   context: {
      name: prompt("What is your name?", ""),
      email: prompt("What is your email address?", "")
   }
});
```

1.data:

- Specifies the data to be sent to the server. Can be a query string or a dictionary object.
- Example:

```
data: "query=pizza&page=2"
```

2. dataType:

- Specifies the type of data expected from the server: `html`, `json`, `jsonp`, `script`, `text`, `xml`.
- Example:

```
$.ajaxSetup({ dataType: "text" });
```

3.error:

- A callback function executed if the request fails.
- Example:

```
$.ajax({ error: function(XMLHttpRequest, textStatus, errorThrown) {
   registerError(textStatus);
}});
```

4.success:

- A callback function executed if the request succeeds.
- Example:

```
$.ajax({ success: function(data, textStatus, XMLHttpRequest) {
    processData(data);
}});
```

5.type:

- Specifies the HTTP method to use (`GET`, `POST`, etc.). The default is `GET`.
- Example:

```
type: "POST"
```

6. url:

- The URL to which the request is sent.
- Example:

```
url: "/update-user"
```

$.ajaxSetup()

- Allows setting default values for future Ajax requests, reducing repetition.
- Example:

```
$.ajaxSetup({ dataType: "text", type: "POST" });
```

Convenience Methods: $.get() and $.post()

- Simplified versions of `$.ajax()` for common `GET` and `POST` requests.

Sample Invocations:

```
$.get("/resources/update");

$.post("/resources/update", { user: "jsmith", product_id: 112 });
```

JQuery

Introduction

jQuery is a popular JavaScript library that simplifies DOM manipulation, event handling, and AJAX interactions. It provides a rich set of functions and selectors that allow developers to write concise and powerful code, making it a virtual higher-level language for JavaScript. This guide will explore various jQuery selectors and demonstrate how they can be combined to create complex queries and manipulate the DOM efficiently.

Selectors

Selectors in jQuery are powerful tools that allow you to select and manipulate HTML elements based on various criteria. Here's a detailed look at some common selectors:

1. **Basic Selectors**:
   - $("*"): Selects all DOM elements.
   - $("#id"): Selects Id attribute.
   - $(".class"): Selects class attribute .
2. **Attribute Selectors**:
   - $("id!=result"): Matches elements with id not equal to result.
   - $("class^=main"): Matches elements with a class starting with main.
3. **Form Selectors**:
   - $(":button"): Selects all buttons.
   - $(":checkbox"): Selects all checkboxes.
   - $(":checked"): Selects all checked inputs.
   - $(":disabled"): Selects all disabled inputs.
   - $(":enabled"): Selects all enabled elements.
   - $(":file"): Selects all file inputs.
   - $(":password"): Selects all password inputs.
   - $(":radio"): Selects all radio inputs.
   - $(":reset"): Selects all reset inputs.
   - $(":submit"): Selects all submit inputs.
   - $(":text"): Selects all text inputs.
4. **Child Selectors**:
   - $("p").eq(0): Selects the first p element.
   - $(":even"): Selects all even-numbered elements (zero-based).

  - $(":first-child"): Selects the first child of each parent.
  - $(":gt(2)"): Selects elements with index greater than 2.
  - $(":has(a)"): Selects elements containing an a element.
  - $(":header"): Selects all header elements (h1, h2, etc.).
  - $(":hidden"): Selects all hidden elements.
  - $(":image"): Selects all images.
  - $(":input"): Selects all form inputs.
  - $(":last-child"): Selects the last child of each parent.
  - $(":last"): Selects the last element in a set.
  - $(":lt(2)"): Selects elements with index less than 2.
  - $(":not(p a)"): Selects p elements that do not contain a elements.
  - $(":nth-child(3n)"): Selects every third child.
  - $(":odd"): Selects all odd-numbered elements (zero-based).
  - $(":only-child"): Selects elements that are the only child of their parent.
  - $(":parent"): Selects elements that are parents of other elements.
  - $(":selected"): Selects all selected elements.
  - $(":visible"): Selects all visible elements.

Combining Selectors

Selectors can be combined to create complex queries. For example:

- $("div.product ul:nth-child(odd)"): Selects the first, third, fifth, etc., ul elements inside a div with the class product.

1. EXPLAIN XHTML HTTP REQUEST AND RESPONSE

HTTP Request

An HTTP request is a message sent by a client to a server to initiate an action or retrieve data. In the context of web development, this often involves a web browser (the client) requesting resources or services from a web server. The request message typically includes several key components:

1. **Request Line**: This specifies the HTTP method (such as GET, POST, PUT, DELETE), the URL of the resource, and the HTTP version.
   - Example: GET /index.html HTTP/1.1
2. **Headers**: These provide additional information about the request, such as the type of data the client can accept (Accept header), the host (Host header), and user agent details (User-Agent header).

EX:

```
Host: www.example.com

User-Agent: Mozilla/5.0

Accept: text/html
```

3. **Body**: This is optional and typically included in methods like POST and PUT, where the client sends data to the server. The body contains the actual data being transmitted.

   - Example: For a POST request submitting form data

     ```
     username=johndoe&password=secret
     ```

HTTP Response

An HTTP response is the message a server sends back to the client in response to an HTTP request. The response message also consists of several key components:

1. **Status Line**: This includes the HTTP version, a status code indicating the result of the request, and a brief reason phrase.
   - Example: HTTP/1.1 200 OK
2. **Headers**: These provide metadata about the response, such as the content type (Content-Type header), length of the response body (Content-Length header), and caching policies (Cache-Control header).
   - Example:

     ```
     Content-Type: text/html

     Content-Length: 342
     ```
3. **Body**: This contains the actual data requested by the client, such as HTML, JSON, or an image.

   - Example: An HTML document returned in the body:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Example</title>
```

```
</head>
<body>
  <h1>Hello, world!</h1>
</body>
</html>
```

The Request-Response Cycle

1. **Client Initiates Request**: When you type a URL in your browser and press Enter, the browser creates an HTTP request and sends it to the server specified in the URL.
2. **Server Processes Request**: The server receives the request, processes it (e.g., fetching data from a database, generating HTML), and prepares an HTTP response.
3. **Server Sends Response**: The server sends the HTTP response back to the client, containing the requested resource or data.
4. **Client Receives Response**: The client (browser) receives the response and renders the content (e.g., displays the HTML page, processes JSON data for JavaScript).

2. Develop a search application in Django using AJAX that displays courses enrolled by a student being searched.

Step 1: Create Django Project and create Search app as we did in earlier modules

Step2:Create Models for database

```
from django.db import models
class Course(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Student(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
    courses = models.ManyToManyField(Course)

    def __str__(self):
        return self.name
```

Apply Migrations **python manage.py makemigrations**

python manage.py migrate

Step 3: Create View in view.py

```
from django.shortcuts import render
from django.http import JsonResponse
from .models import Student
def search_student(request):
    query = request.GET.get('query', '')
    if query:
        students = Student.objects.filter(name__icontains=query).select_related('courses').all()
        results = [
            {
                'name': student.name,
                'email': student.email,
                'courses': [course.name for course in student.courses.all()],
            } for student in students
        ]
    else:
        results = []
    return JsonResponse({'results': results})

def search_page(request):
    return render(request, 'search/search.html')
```

Step 4: Make URL configuration in both Main Project and app

Step 5: Create template

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Course Search</title>
</head>
<body>

    <h1>Search Courses by Student</h1>
    <input type="text" id="searchInput" placeholder="Enter student name">
    <button id="searchButton">Search</button>

    <div id="results"></div>

    <script>
        document.getElementById('searchButton').addEventListener('click', function() {
            const query = document.getElementById('searchInput').value;
            fetch(`/search/search/?query=${query}`, {
                method: 'GET',
                headers: {
                    'X-Requested-With': 'XMLHttpRequest',
                },
            })
            .then(response => response.json())
            .then(data => {
                const resultsDiv = document.getElementById('results');
                resultsDiv.innerHTML = '';
                if (data.results.length > 0) {
                    data.results.forEach(result => {
                        const studentDiv = document.createElement('div');
                        studentDiv.innerHTML = `<h3>${result.name}
(${result.email})</h3><ul>${result.courses.map(course => `<li>${course}</li>`).join('')}</ul>`;
                        resultsDiv.appendChild(studentDiv);
                    });
                } else {
                    resultsDiv.innerHTML = '<p>No results found</p>';
                }
            })
            .catch(error => {
                document.getElementById('results').textContent = 'An error occurred. Please try again.';
            });
        });
    </script>
</body>
</html>
```

Run the server python manage.py runserver