server prog. mid questions

//winter mid 2019-2020
2.a) There are four layers of web service technology stack. Mention the responsibility of each layer with one of the mechanisms or standards to implement them.
b) Suppose you are going to develop an application named Stack Underflow where users can post their blog articles. You will make a RESTful web service for that. Now you will let the user to request to your web service to add, edit, delete, get all the articles and get a single article through your service.
What will be the URLs and HTTP verbs for the requests? Mention which URLs will be used for which purposes.

Memory updated

2.a) Four Layers of Web Service Technology Stack:

1. **Presentation Layer:**
   - **Responsibility:** Handles user interface and client-side logic. It communicates with the service layer.
   - **Mechanism/Standard:** HTML, CSS, and JavaScript frameworks like React or Angular.
2. **Service Layer:**
   - **Responsibility:** Manages the business logic and processes client requests.
   - **Mechanism/Standard:** RESTful web services, SOAP.
3. **Data Access Layer:**
   - **Responsibility:** Manages access to the database and performs CRUD (Create, Read, Update, Delete) operations.
   - **Mechanism/Standard:** JDBC, Hibernate, or JPA for database interaction.
4. **Database Layer:**
   - **Responsibility:** Stores and retrieves the application's data.
   - **Mechanism/Standard:** SQL databases (e.g., MySQL, PostgreSQL) or NoSQL databases (e.g., MongoDB).

2.b) URLs and HTTP Verbs for RESTful Web Service in *Stack Underflow*:

1. **Add a New Article:**
   - **URL:** `` `POST /articles` ``
   - **HTTP Verb:** `` `POST` ``
   - **Purpose:** Used to create and add a new blog article.

2. **Edit an Existing Article:**
    - **URL:** `` `PUT /articles/{id}` ``
    - **HTTP Verb:** `` `PUT` ``
    - **Purpose:** Used to update an existing article using its unique identifier (`` `{id}` ``).
3. **Delete an Article:**
    - **URL:** `` `DELETE /articles/{id}` ``
    - **HTTP Verb:** `` `DELETE` ``
    - **Purpose:** Used to delete an article based on its unique identifier (`` `{id}` ``).
4. **Get All Articles:**
    - **URL:** `` `GET /articles` ``
    - **HTTP Verb:** `` `GET` ``
    - **Purpose:** Used to retrieve a list of all articles.
5. **Get a Single Article:**
    - **URL:** `` `GET /articles/{id}` ``
    - **HTTP Verb:** `` `GET` ``
    - **Purpose:** Used to retrieve a single article by its unique identifier (`` `{id}` ``).

1.a) Two samples of HTTP request message and HTTP response message are given in figure 1 and 2 respectively. Explain the format of the messages in the figures. Explain each of the terms in no more than two sentences.
POST/login HTTP/1.1
Host: www.stackunderflow.com
Content-Type: application/json
Content-Length: xx
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
{"username":"myusername", "password": "mypassword"}
//this was Figure 1: Sample HTTP request message for question 1.a
HTTP/1.1 409 Conflict
Content-Type: application/json
Content-Length: xx
{"message": "Invalid username or password"}
//this was Figure 2: Sample HTTP response message for question 1.a
b) Suppose you have requested a server that has given you response with status code in one of the following formats. What will you understand by the following status codes? Explain each in one sentence.
i. 2xx
ii. 4xx
5xx
c) In which context the CONNECT HTTP method is used?

1.a) Explanation of the HTTP Request (Figure 1):

- **POST/login HTTP/1.1:** This is the request line. `POST` indicates the HTTP method used, `/login` is the resource being accessed, and `HTTP/1.1` is the version of the HTTP protocol.
- **Host:** www.stackunderflow.com**:** Specifies the server to which the request is being sent.
- **Content-Type: application/json:** Indicates the media type of the resource being sent in the body, in this case, JSON.
- **Content-Length: xx:** Indicates the size of the body content being sent (represented by `xx` here).
- **User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT):** Identifies the client making the request, including the browser and operating system.
- **Accept-Language: en-us:** Specifies the preferred language for the response.
- **Accept-Encoding: gzip, deflate:** Informs the server of the accepted encoding formats for the response.
- **Connection: Keep-Alive:** Requests that the server keeps the connection open after the transaction, allowing further requests.
- **{"username":"myusername", "password": "mypassword"}:** This is the message body, containing the data being sent (in JSON format) as part of the request.

Explanation of the HTTP Response (Figure 2):

- **HTTP/1.1 409 Conflict:** This is the status line. `HTTP/1.1` is the HTTP version, and `409 Conflict` is the status code indicating a conflict (in this case, an invalid login).
- **Content-Type: application/json:** Specifies the format of the response body (JSON in this case).
- **Content-Length: xx:** Indicates the size of the body content.
- **{"message": "Invalid username or password"}:** This is the message body, containing the error message in JSON format.

1.b) Explanation of Status Codes:

i. **2xx (Success):** Indicates that the request was successfully received, understood, and accepted by the server. Common examples are `200 OK` and `201 Created`.

ii. **4xx (Client Error):** Indicates that there was an error with the request, usually due to incorrect input from the client (e.g., `404 Not Found`, `400 Bad Request`).

iii. **5xx (Server Error):** Indicates that the server encountered an error while trying to process the request (e.g., `500 Internal Server Error`, `503 Service Unavailable`).

1.c) CONNECT HTTP Method:

The **CONNECT** method is used to establish a tunnel to the server identified by the target resource. It is typically used with proxies to allow a client to establish a two-way connection for SSL (HTTPS) connections.

//winter mid 2022-2023
1.Imagine you want to build a new social media platform "connectpal". Here users can connect with each other based on common interests, share their piece of mind with others, and also play group quests with their friends. Answer the following concerns regarding the platform.
a) In the platform, you want to implement HTTP for transferring files - such as text, images, sound, video, and other multimedia files over the web. Defend your decision as to why you would choose HTTP over different protocols.

b) Provide an example for the following types of HTTP request methods to design the backend infrastructure for "ConnectPal":
GET

HEAD
POST
PUT
DELETE
CONNECT
OPTIONS
TRACE
PATCH

c)You have created the following URL. that will be used to access a specific group quest page on "ConnectPal", which will allow users to engage in collaborative gaming activities with others who share the same interest in gaming. Now explain different parts of the following URL-
https://www.connectpal.com:101/groups/quest?id=123&interests=RPG+gaming#level1
d)When users click on the given URL in Question 1(c), how will you establish the connection between the client and server? Describe the HTTP Request and Response format for the given URL in Question 1(c).

1.a) Why Choose HTTP for Transferring Files in "ConnectPal":

HTTP is a widely adopted protocol that allows efficient transfer of text, images, sound, video, and other multimedia files. Key reasons for choosing HTTP over other protocols:

- **Ubiquity and Compatibility:** HTTP is universally supported across all web browsers and devices, making it highly compatible for users accessing ConnectPal.
- **Statelessness:** HTTP is stateless, meaning each request is independent, which simplifies the scalability of the platform.
- **Secure Transfer (HTTPS):** Using HTTPS ensures secure data transmission over the network, safeguarding user privacy and data integrity.
- **Caching and Optimization:** HTTP supports caching, enabling faster delivery of static files like images and videos.
- **Flexible Content Delivery:** It supports various content types and encodings, allowing efficient delivery of multimedia files.

1.b) Example of HTTP Request Methods for "ConnectPal" Backend:

1. **GET** – Retrieve user profile information:
   - **Example:** `GET /users/profile?id=45`
   - **Purpose:** To fetch the user's profile details.

2. **HEAD** – Check if a user exists without downloading the content:
   - **Example:** `HEAD /users/profile?id=45`
   - **Purpose:** Retrieve only the headers to verify user existence.

3. **POST** – Create a new post on the timeline:
   - **Example:** `POST /posts`
   - **Purpose:** To submit a new post with content like text or images.

4. **PUT** – Update user profile information:
   - **Example:** `PUT /users/profile?id=45`
   - **Purpose:** To modify an existing user's profile details.

5. **DELETE** – Delete a user post:
   - **Example:** `DELETE /posts?id=234`
   - **Purpose:** To remove a specific post from the timeline.

6. **CONNECT** – Set up a tunnel to a gaming server for real-time interaction:
   - **Example:** `CONNECT /game/server`
   - **Purpose:** Establish a direct connection for multiplayer gaming.

7. **OPTIONS** – Retrieve communication options for user settings:
   - **Example:** `OPTIONS /users/settings`
   - **Purpose:** To get the available methods and options supported for user settings.

8. **TRACE** – Debug connection issues between client and server:
   - **Example:** `TRACE /users/connection`
   - **Purpose:** To trace the route of a request from the client to the server.

9. **PATCH** – Modify a specific field in a user profile:
   - **Example:** `PATCH /users/profile?id=45`
   - **Purpose:** To update a specific detail, such as changing the profile picture.

1.c) Breakdown of the URL:

URL: `https://www.connectpal.com:101/groups/quest?id=123&interests=RPG+gaming#level1`

- **https://**: The protocol used (HTTPS) ensures secure communication between the client and server.
- www.connectpal.com: The domain name, which identifies the server hosting the ConnectPal application.
- **:101**: The port number, which specifies the server's port to connect to (here, it's port 101 instead of the default port 443 for HTTPS).
- **/groups/quest**: The path indicating the specific resource (in this case, the group quest page).

- **?id=123&interests=RPG+gaming**: The query string, containing parameters (`id` and `interests`) to filter or identify the specific quest.
  - **id=123**: Refers to the unique ID of the group quest.
  - **interests=RPG+gaming**: Refers to the user's interest category (RPG gaming).
- **#level1**: The fragment identifier, which specifies a section (level 1) of the page to navigate to.

1.d) Establishing the Connection and HTTP Request/Response Format:

When a user clicks on the URL, the browser sends an HTTP request to the server at `www.connectpal.com` on port 101, requesting the resource `/groups/quest?id=123&interests=RPG+gaming#level1`. The server processes the request and responds with the corresponding content.

HTTP Request:

```makefile
GET /groups/quest?id=123&interests=RPG+gaming HTTP/1.1
Host: www.connectpal.com:101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html,application/xhtml+xml
Accept-Language: en-us
Accept-Encoding: gzip, deflate, br
Connection: Keep-Alive
```

- **GET /groups/quest?id=123&interests=RPG+gaming HTTP/1.1:** The client is making a GET request to the specified path with parameters.
- **Host:** Specifies the server and port.
- **User-Agent:** Identifies the browser making the request.
- **Accept:** Specifies the content types that the client can handle.
- **Accept-Language/Encoding:** Specifies preferences for language and compression.
- **Connection:** Indicates the desire to keep the connection alive for future requests.

HTTP Response:

```less
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1234
Connection: Keep-Alive

<html>
  <body>
    <h1>Welcome to RPG Group Quest Level 1!</h1>
    ...
  </body>
</html>
```

- **HTTP/1.1 200 OK:** The server indicates that the request was successful.
- **Content-Type:** Specifies that the response is HTML.
- **Content-Length:** Specifies the size of the response content.

- **Connection:** The server indicates that the connection can remain open.
- **Body:** Contains the HTML content for the requested group quest page.

2. a)Imagine you have started your career at "MetaLow" as a backend developer. Your company is developing a web application for a healthcare system that needs to exchange patient in formation securely between various medical facilities. As a backend developer what type of web service will you choose? Justify your answer.

b)Suppose you are working on a project to integrate weather information into a mobile weather app WeatherMatic. You have two options: using a web service or integrating with a weather API. Justify your preferred option.
(c) If you want to implement a proxy server for the WeatherMatic app mentioned in Question 2(b), what type of proxy server can you implement and where can you implement it? What are the advantages and disadvantages of your decision?

2.a) Choosing a Web Service for Secure Healthcare Information Exchange at "MetaLow":

I would choose **SOAP (Simple Object Access Protocol)** over REST for this healthcare system.

Justification:

- **Security:** SOAP is better suited for secure transactions, especially in healthcare, as it supports WS-Security, ensuring message integrity and confidentiality. It complies with security standards like HIPAA (Health Insurance Portability and Accountability Act), which is crucial for exchanging sensitive patient data.
- **Reliability:** SOAP has built-in support for ACID-compliant transactions, ensuring reliable communication between different medical facilities where data loss or inconsistency can't be tolerated.
- **Standardization:** SOAP offers formal contracts via WSDL (Web Services Description Language), which are essential for interoperability between different healthcare systems.

2.b) Using a Web Service vs. Weather API for WeatherMatic:

I would prefer integrating **with a Weather API** rather than using a custom web service.

Justification:

- **Simplicity:** Most weather APIs provide straightforward endpoints that return the needed weather data in standard formats (like JSON or XML), minimizing the development effort.
- **Real-time Data:** Weather APIs are typically maintained by third-party providers with access to real-time weather data from various sources, ensuring data accuracy and availability.
- **Cost and Maintenance:** Using an API is often more cost-effective, as developing and maintaining a custom web service for weather data would require significant resources. Leveraging an API reduces this overhead.

- **Focus on Core Functionality:** Integrating a third-party weather API allows the development team to focus on enhancing the app's core functionality without having to manage the complexities of sourcing, aggregating, and updating weather data.

2.c) Proxy Server for WeatherMatic:

I would implement a **forward proxy server** for the WeatherMatic app.

Location of Implementation:

- The forward proxy server would sit between the mobile weather app and external weather APIs. It would be placed in the application's infrastructure, handling all outgoing requests to the API.

Advantages:

- **Caching:** The proxy server can cache weather data for a short period, reducing the number of API requests and improving response time for users, especially for repeated requests for the same location.
- **Security and Privacy:** By using a proxy, the mobile app's backend can hide its IP address, adding a layer of privacy for users and preventing direct exposure to external API servers.
- **Traffic Control:** It can help manage traffic, load balancing between API requests, and enforce rate limiting, ensuring that the app does not exceed the API provider's usage limits.

Disadvantages:

- **Latency:** Introducing a proxy adds an additional layer in the communication flow, potentially increasing latency, especially if the proxy becomes a bottleneck.
- **Complexity and Maintenance:** Setting up and maintaining a proxy server requires additional infrastructure, increasing operational complexity and costs, as well as monitoring for failures or performance issues.