Qst: What is Flask ?
Ans: Flask is a lightweight and flexible web framework for Python. It is designed to be simple and easy to use, providing the essentials for building web applications without imposing too many constraints. Flask is known for its simplicity, extensibility, and ease of integration with other libraries and tools. It follows the WSGI (Web Server Gateway Interface) standard, making it compatible with various web servers.

Qst: What is Flask-SqlAlchemy?
Ans: Flask-SQLAlchemy is an extension for Flask that integrates SQLAlchemy with Flask. SQLAlchemy is a powerful and flexible Object-Relational Mapping (ORM) library for Python, and Flask-SQLAlchemy simplifies its use in Flask applications by providing useful abstractions and integrating it seamlessly into the Flask ecosystem.

Qst:- What is current_app in flask?
Ans: In Flask, `current_app` is a proxy for the application context, which allows you to access the application instance from anywhere within the Flask application, especially outside of view functions and request contexts. `current_app` is a powerful feature in Flask that allows you to access the Flask application instance from various parts of your code, as long as you're within the application context.

Qst:- What is the context in flask?
Ans: In Flask, the concept of "context" is central to how the application manages and provides access to various components like the application instance, request data, and more. Contexts in Flask help manage state across different parts of the application, ensuring that certain information is available where it's needed without having to pass it around explicitly.

Here's a breakdown of the different types of contexts in Flask:

1. Application Context

The application context (`app_context`) is used to manage information about the Flask application, such as configuration settings and application-level utilities. When the application context is pushed, Flask makes the application instance available through `current_app` and `g`.

- **`current_app`**: This proxy allows you to access the current application instance.
- **`g`**: This is a global context object for storing data during a request. It can be used to store data that you want to be available throughout the lifetime of the request.

2. Request Context

The request context (`request_context`) is used to manage data specific to a request, such as request headers, form data, and session data. When the request context is pushed, Flask makes the request object available through `request`, and also provides access to `session`.

- `request`: This object contains data about the current HTTP request.
- `session`: This object allows you to store data that persists across requests for a particular user.

Qst:-What id db.model?

Ans: `db.Model`: In Flask-SQLAlchemy, `db.Model` is the base class for all models that interact with the database. It provides a set of features and methods that help manage the relationship between Python classes and the database tables.

Qst:-What is ORM?

Ans: ORM, or Object-Relational Mapping, is a programming technique used to interact with relational databases using object-oriented programming principles. It abstracts the database interactions into high-level objects, allowing developers to work with database data using objects rather than raw SQL queries

Key Concepts of ORM

1. **Mapping Objects to Tables**: ORM tools map classes (objects) to database tables. Each instance of a class corresponds to a row in the table, and each attribute of the class corresponds to a column in the table.
2. **Automatic SQL Generation**: ORMs handle the conversion between object-oriented code and SQL queries. This means you don't need to write raw SQL; instead, you work with objects, and the ORM generates the appropriate SQL commands behind the scenes.
3. **CRUD Operations**: ORMs simplify Create, Read, Update, and Delete operations. Instead of writing SQL queries, you use methods provided by the ORM to perform these operations.
4. **Relationships**: ORMs manage relationships between objects (e.g., one-to-many, many-to-many) and automatically handle the necessary SQL joins and constraints.

Qst:-What is the difference between backref and backpopulate?

Ans: In ORMs, such as SQLAlchemy, `backref` and `back_populates` are mechanisms used to manage relationships between models. They help in synchronizing the bidirectional relationship between two models.

`backref`

- **Purpose**: Creates a bidirectional relationship with a single line of code.
- **Usage**: You use `backref` when you want to define a two-way relationship between models and prefer a shorthand approach.
- **Automatic Synchronization**: It automatically creates a complementary property on the related model, which behaves like a normal attribute but also maintains the relationship in both directions.

`back_populates`

- **Purpose**: Provides explicit control over the bidirectional relationship.
- **Usage**: Use `back_populates` when you want to clearly define the relationships on both sides of the relationship with explicit mapping.
- **Manual Synchronization**: You manually specify which attribute on each side of the relationship should be used to synchronize the relationship.

Qst: declarative_base vs db.model ?

Ans: In SQLAlchemy and its extensions like Flask-SQLAlchemy, both `declarative_base` and `db.Model` are used to define models, but they are part of different contexts and serve slightly different purposes. Here's a breakdown of the differences between the two:

1. `declarative_base` (SQLAlchemy Core)

- **Context**: `declarative_base` is part of the core SQLAlchemy library and is used in pure SQLAlchemy applications, not necessarily tied to Flask.
- **Purpose**: It creates a base class that your models inherit from. This base class is responsible for mapping model classes to database tables using SQLAlchemy's declarative system.

2. `db.Model` (Flask-SQLAlchemy)

- **Context**: `db.Model` is provided by Flask-SQLAlchemy, an extension for Flask that integrates SQLAlchemy with Flask applications.
- **Purpose**: It simplifies the use of SQLAlchemy within Flask applications. `db.Model` is a subclass of `sqlalchemy.ext.declarative.api.DeclarativeMeta`, which makes it easier to define models and interact with the database in a Flask-friendly way.

Qst: What is @app.route ?

Ans: In Flask, the @app.route decorator is used to define routes for your web application. A route maps a URL path to a function, allowing you to handle web requests and send responses.

Qst: What is Template inheritance ?

Ans: Template inheritance is a key feature in modern web development frameworks and templating engines that allows you to create a base template with common structure and layout, and then extend or override parts of it in child templates. This helps in maintaining a consistent look and feel across multiple pages while allowing for customization and flexibility.

Qst:- What is MVC?

Ans: MVC stands for Model-View-Controller. It's a design pattern commonly used in software development to separate concerns and organize code

Here's a brief overview of each component:

1. **Model**: This represents the data and the business logic of the application. It is responsible for retrieving, storing, and processing data. In a web application, this could be the code that interacts with the database.
2. **View**: The view is responsible for rendering the user interface and displaying data from the model. It presents the data to the user and sends user interactions to the controller. Essentially, it is what the user sees.
3. **Controller**: The controller acts as an intermediary between the model and the view. It receives user input from the view, processes it (often by updating the model), and then updates the view accordingly.

Qst:- url_for vs render_template ?

Ans: **url_for** generates a URL for a given function or endpoint.
It's typically used to create URLs dynamically, ensuring that your links remain consistent even if routes or URLs change.
render_template renders an HTML template and returns it as a response
It's used to combine data with HTML templates to generate a dynamic web page.

Qst: What is API and RESTful API ?

Ans: API-An API (Application Programming Interface) **is a set of rules and protocols that allows different software applications to communicate with each other**. It defines methods and data formats for interacting with a service or component, enabling developers to integrate and extend functionalities without needing to understand the internal workings of the system they're interacting with.
A RESTful API (Representational State Transfer API) is a web service design pattern that provides a way **to access and manipulate resources over HTTP.** It uses standard HTTP methods and status codes to create, read, update, and delete resources. RESTful APIs are widely used because they are stateless, scalable, and simple to integrate with various clients.

Key Concepts of RESTful APIs

1. **Resources**:
   - Resources are entities or objects that the API manages. Each resource is identified by a URL.
   - For example, in a blog API, `https://api.example.com/posts` might represent a collection of blog posts, while `https://api.example.com/posts/123` represents a specific post with ID 123.
2. **HTTP Methods**:
   - **GET**: Retrieve information about a resource.
   - **POST**: Create a new resource.
   - **PUT**: Update an existing resource.
   - **PATCH**: Apply partial modifications to a resource.

    - **DELETE**: Remove a resource.
3. **Stateless**:
    - Each request from a client to the server must contain all the information needed to understand and process the request. The server does not store any information about the client between requests.
4. **Uniform Interface**:
    - REST APIs have a consistent and predictable interface. The interaction between clients and servers is standardized.
5. **Representation**:
    - Resources can be represented in various formats such as JSON, XML, or HTML. JSON is the most common format used in modern RESTful APIs.
6. **Statelessness**:
    - The API should not rely on server-side sessions or state. Each request from the client must contain all the necessary information.
7. **Use of HTTP Status Codes**:
    - Status codes provide information about the result of the API request. For example:
        - `200 OK` for successful requests.
        - `201 Created` when a resource is successfully created.
        - `204 No Content` for successful requests that do not return any content.
        - `400 Bad Request` for invalid request syntax or parameters.
        - `404 Not Found` if the requested resource is not found.
        - `500 Internal Server Error` for server-side issues.

Qst: Framework vs Libraries ?
Ans:
Framework- A framework is a comprehensive platform or set of tools designed to support the development of a specific type of application. It provides a structured foundation with pre-defined conventions, patterns, and components that guide the development process.

Library: A library is a collection of pre-written code that developers can use to perform common tasks or add specific features to their applications. Unlike a framework, a library does not dictate the overall structure or flow of your application.

Qst: What is Normalisation ?
Ans: Normalization in the context of databases **is the process of organizing data to reduce redundancy and improve data integrity**. It involves decomposing a database into smaller, more manageable tables and defining relationships between them.

Qst: ACID vs BASE ?
Ans: ACID stands for Atomicity, Consistency, Isolation, and Durability. It represents a set of properties that guarantee that database transactions are processed reliably and ensure the integrity of the database.

1. **Atomicity**:
    - **Definition**: Ensures that a transaction is treated as a single, indivisible unit. Either all operations within the transaction are completed successfully, or none are. If an error occurs, the transaction is rolled back to its initial state.
    - **Example**: In a bank transfer, both the debit from one account and the credit to another must succeed together. If one fails, both operations are rolled back.
2. **Consistency**:
    - **Definition**: Ensures that a transaction takes the database from one valid state to another valid state. The database must adhere to all predefined rules, constraints, and triggers.
    - **Example**: If a database enforces a constraint that an account balance cannot go below zero, consistency ensures this constraint is maintained after every transaction.
3. **Isolation**:
    - **Definition**: Ensures that transactions are executed independently of one another. Intermediate results of a transaction are not visible to other transactions until the transaction is complete.
    - **Example**: Two transactions updating the same record should not interfere with each other's operations. One transaction's changes should not be visible to another until both transactions are committed.
4. **Durability**:
    - **Definition**: Ensures that once a transaction is committed, its effects are permanent and survive any subsequent system failures.
    - **Example**: After a successful transaction to update a record, the changes must be stored persistently, even if there is a system crash.

ACID Properties are typically associated with traditional relational databases (RDBMS) like MySQL, PostgreSQL, and Oracle, where maintaining data integrity and reliability is crucial.

BASE stands for **Basically Available, Soft state,** and **Eventually consistent**. It represents a different set of properties more commonly associated with distributed databases and NoSQL systems.

1. **Basically Available**:
    - **Definition**: Ensures that the system is generally available, meaning that it can respond to requests most of the time. However, it does not guarantee that all data will be up-to-date or consistent at all times.
    - **Example**: In a distributed system, some nodes may be temporarily out of sync, but the system as a whole will continue to operate and handle requests.
2. **Soft State**:
    - **Definition**: Indicates that the state of the system may change over time, even without new input. This is because data consistency might not be immediate.
    - **Example**: Updates to a database might propagate gradually across nodes, meaning that the state of the database can change as updates are applied.
3. **Eventually Consistent**:

- **Definition**: Ensures that, given enough time, all updates will propagate through the system and all nodes will eventually become consistent. However, consistency is not guaranteed at any specific point in time.
- **Example**: A distributed database may have some nodes showing outdated data temporarily, but eventually, all nodes will reflect the most recent changes.

BASE Properties are typically associated with NoSQL databases like Cassandra, DynamoDB, and Couchbase, where scalability and availability are prioritized, and strict consistency is relaxed in favor of better performance and fault tolerance.

Qst: RDBMS vs NoSQL ?

Ans: Relational Databases (RDBMS) are a type of database management system that organizes data into tables, which are structured in rows and columns. Each table represents a specific entity (such as customers or orders), and relationships between these entities are defined through primary and foreign keys. RDBMSs use Structured Query Language (SQL) for querying and managing data.

Key Features of RDBMS

1. **Structured Data**:
   - Data is organized in tables (relations) with rows and columns. Each row represents a record, and each column represents an attribute of the record.
2. **Schema**:
   - **Fixed Schema**: The structure of the tables, including the data types and constraints, is defined in advance. Changing the schema requires altering the database structure.
3. **Data Integrity**:
   - **ACID Properties**: RDBMSs ensure that transactions are processed reliably through ACID properties (Atomicity, Consistency, Isolation, Durability).
     - **Atomicity**: Transactions are all-or-nothing.
     - **Consistency**: Transactions bring the database from one valid state to another.
     - **Isolation**: Transactions are executed independently.
     - **Durability**: Once committed, changes are permanent.
4. **Relationships**:
   - **Primary Keys**: Uniquely identify each record in a table.
   - **Foreign Keys**: Establish relationships between tables by referencing primary keys in other tables.
5. **Query Language**:
   - **SQL (Structured Query Language)**: Used for defining, manipulating, and querying data. SQL allows complex queries, including joins, aggregations, and transactions.
6. **Data Normalization**:

    - **Normalization**: The process of organizing data to reduce redundancy and improve data integrity. It involves dividing large tables into smaller, related tables and defining relationships.
7. **Transactions**:
    - Support for complex transactions that involve multiple operations, ensuring data integrity and consistency.

Examples of RDBMS

1. **MySQL**:
    - Open-source RDBMS known for its ease of use and performance. Widely used in web applications and small to medium-sized applications.
2. **PostgreSQL**:
    - Open-source RDBMS with advanced features and strong SQL compliance. Known for its extensibility and support for complex queries.
3. **Oracle Database**:
    - Commercial RDBMS known for its robustness, scalability, and enterprise features. Often used in large-scale, mission-critical applications.
4. **Microsoft SQL Server**:
    - Commercial RDBMS developed by Microsoft. Known for its integration with other Microsoft products and support for various data management features.
5. **SQLite**:
    - A lightweight, file-based RDBMS used in mobile applications, embedded systems, and small-scale applications.

NoSQL (Not Only SQL) databases are a broad class of database management systems designed to handle a wide variety of data models, including structured, semi-structured, and unstructured data. They offer alternatives to the traditional relational database model, focusing on scalability, flexibility, and performance. Here's a comprehensive overview of NoSQL databases

Key Features of NoSQL Databases

1. **Flexible Schema**:
    - **Schema-Less**: Many NoSQL databases do not require a fixed schema. This allows for dynamic changes in the structure of the data, making it easier to store varied and evolving data formats.
2. **Scalability**:
    - **Horizontal Scaling**: NoSQL databases are often designed for horizontal scaling, which means they can distribute data across multiple servers to handle increased load and large datasets. This contrasts with the vertical scaling of traditional RDBMSs, which involves adding more resources to a single server.
3. **Varied Data Models**:

    - NoSQL databases support multiple data models to suit different types of data and application requirements:
        - **Document-Based**: Stores data in documents (e.g., JSON, BSON). Examples: MongoDB, CouchDB.
        - **Key-Value Stores**: Stores data as key-value pairs. Examples: Redis, DynamoDB.
        - **Column-Family Stores**: Stores data in columns rather than rows. Examples: Apache Cassandra, HBase.
        - **Graph Databases**: Stores data as nodes and edges, optimized for representing and querying relationships. Examples: Neo4j, Amazon Neptune.
4. **Eventual Consistency**:
    - Many NoSQL databases prioritize availability and partition tolerance (BASE properties) over strict consistency (ACID properties). This means that while data may not be immediately consistent across all nodes, it will eventually become consistent.
5. **High Availability**:
    - Designed to be highly available and fault-tolerant, ensuring that the system remains operational even in the event of hardware failures or network issues.
6. **Performance**:
    - Often optimized for high-speed read and write operations, particularly for large-scale and high-traffic applications.

Types of NoSQL Databases

1. Document-Based Databases

- **Description**: Store and manage data in flexible, hierarchical documents (e.g., JSON, BSON). Each document can contain nested data and varied structures.
- **Examples**:
    - **MongoDB**: Known for its flexible schema and powerful query capabilities. Data is stored in BSON (Binary JSON) format.
    - **CouchDB**: Uses JSON for data storage and JavaScript for MapReduce queries. Known for its ease of replication and synchronization.

2. Key-Value Stores

- **Description**: Store data as key-value pairs. Ideal for simple data retrieval and caching.
- **Examples**:
    - **Redis**: An in-memory data structure store used as a database, cache, and message broker. Supports various data types like strings, lists, sets, and hashes.
    - **DynamoDB**: A fully managed NoSQL database service by AWS that supports key-value and document data models.

3. Column-Family Stores

- **Description**: Store data in columns rather than rows. Suitable for applications requiring efficient read and write operations on large datasets.
- **Examples**:
    - **Apache Cassandra**: Known for its high scalability and distributed architecture. Suitable for applications requiring continuous uptime and handling large amounts of data.
    - **HBase**: Built on top of HDFS (Hadoop Distributed File System) for high-throughput and real-time read/write access.

4. Graph Databases

- **Description**: Store data as graphs, with nodes representing entities and edges representing relationships. Optimized for querying complex relationships and network traversal.
- **Examples**:
    - **Neo4j**: A popular graph database that uses the Cypher query language for querying and analyzing graph data.
    - **Amazon Neptune**: A fully managed graph database service that supports both property graph and RDF graph models.

Qst: Vcs vs Git ?

Ans: VCS-Version Control Systems are tools or methodologies used to track changes to files and directories over time. They manage the history of changes, facilitate collaboration, and allow for rollback to previous versions. There are several types of version control systems, including:

Git is a specific type of distributed version control system (DVCS) created by Linus Torvalds. It is designed to handle everything from small to very large projects with speed and efficiency.

Qst: Client server vs distributed ?

Ans: Client-server architecture is a model **where client devices request services or resources from a central server**. This model is often used in networked applications and can be either simple or complex.

Distributed architecture involves **multiple interconnected systems or nodes** that work together to provide services or process data. Each node can act as both a client and a server, contributing to a more decentralized approach.

Qst: What is load balancing ?

Ans: Load balancing is a technique used to **distribute incoming network traffic or computational workload across multiple servers** or resources to ensure that no single server becomes a bottleneck. This enhances the performance, reliability, and scalability of applications and services.

Qst: What is scaling ?

Ans: Scaling refers to the process of **adjusting the capacity of a system or application** to handle increased or decreased workloads effectively. Scaling is crucial in ensuring that a system remains responsive, reliable, and cost-effective as demands change. There are two primary types of scaling: vertical scaling and horizontal scaling.

Qst: What is CDN ?

Ans: A Content Delivery Network (CDN) is a system of distributed servers designed to deliver web content and other resources (like images, videos, scripts, and stylesheets) more efficiently to users based on their geographical location. The primary goals of a CDN are to improve the performance, reliability, and scalability of delivering content.

Qst: HTTP VS HTTPS ?

Ans: HTTP (**Hypertext Transfer Protocol**) and HTTPS (**Hypertext Transfer Protocol Secure**) are both protocols used for transmitting data over the web. They are fundamental to how information is transferred between a client (such as a web browser) and a server. However, **they differ in terms of security and the mechanisms used to protect data during transmission.**

Qst: Sessions vs Cookies ?

Ans: Sessions are **server-side storage mechanisms** used to keep track of user state and data across multiple requests. The server generates a unique session ID and stores session data, while the client receives and stores this ID in a cookie.
Cookies are small pieces of **data stored on the client's browser**. They are sent to the server with every request made to the domain that set the cookie.

Qst: Authentication vs Authorization ?

Ans: Authentication is the process of **verifying the identity of a user**, system, or entity. It ensures that the entity requesting access to a system is who they claim to be.
Authorization is the process of **determining what an authenticated user or system is allowed to do**. It defines the permissions and access levels for various resources or actions.

Qst: CSS Hierarchy ?

Ans: CSS Hierarchy Summary:
1. Inline Styles:
   - Definition: Styles applied directly to individual HTML elements using the `style` attribute.
   - Hierarchy Note: Inline styles have the highest specificity and override other styles.

2. Internal Styles (Embedded):
   - Definition: Styles defined within the HTML document using the `<style>` tag in the `<head>` section.
   - Hierarchy Note: Internal styles take precedence over external styles but are overridden by inline styles.

3. External Styles:

- Inline styles have the highest specificity and override all other styles.
- ID selectors have higher specificity than class and type selectors.
- Class selectors have higher specificity than type selectors.
- Type selectors have lower specificity compared to ID and class selectors.
- The universal selector has the lowest specificity.

Qst: What is AJAX ?

Ans: AJAX stands for **Asynchronous JavaScript and XML**. It's a technique used in web development to create interactive and dynamic web applications. Here's a breakdown of what it involves:

1. Asynchronous: AJAX allows web pages to update asynchronously by exchanging small amounts of data with the server behind the scenes. This means that parts of a web page can be updated without reloading the entire page.
2. JavaScript: JavaScript is used to send and receive data from the server. It helps manage the data and update the content of a webpage dynamically.
3. XML (and JSON): Originally, XML was used to format the data exchanged between the server and client. However, JSON (JavaScript Object Notation) has become more popular due to its simplicity and ease of use with JavaScript.
4. HTTP Requests: AJAX typically uses the XMLHttpRequest object or the Fetch API to make HTTP requests to the server and retrieve data.

Qst: What are JS frameworks ?

Ans: JavaScript frameworks are pre-written libraries of JavaScript code that provide a structured way to build and manage web applications. They **offer tools and conventions that help developers create dynamic and interactive web applications** more efficiently. Frameworks often include built-in functions, components, and utilities that handle common tasks, so developers don't have to write everything from scratch. **Examples: React,Angular,Vue.Js etc,**