Module 1, 2 and 3

Mod 1,2,3 Qb Answers

Module 4: MongoDB

Q.1 Explain MongoDB Data Types with an example

String	This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
Integer	This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
Boolean	This type is used to store a boolean (true/ false) value.
Double	This type is used to store floating point values.
Min/ Max keys	This type is used to compare a value against the lowest and highest BSON elements.
Arrays	This type is used to store arrays or list or multiple values into one key.
Timestamp	ctimestamp. This can be handy for recording when a document has been modified or added.
Object	This datatype is used for embedded documents.
Null	This type is used to store a Null value.
Symbol	This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
Date	This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.

Object ID	This datatype is used to store the document's ID.
Binary data	This datatype is used to store binary data.
Code	This datatype is used to store JavaScript code into the document.
Regular expression	This datatype is used to store regular expression.

Q.2 Explain the features of MongoDB

- MongoDB is a Database application.
- NoSQL used for high volume storage.
- It is a Collection and Document based.
    - NoSQL database, doesn't follow a schema
    - Documents contain key (field)-value pair
    - Collection is a set of Documents and functions
- Uses BSON to query DB.
- Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.
- The rows (or documents as called in MongoDB) doesn't need to have a schema defined beforehand. Instead, the fields can be created on the fly.
- The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.
- Advantages of MongoDB:

1. **Document-oriented**
    - Since MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very **flexible and adaptable** to real business world situation and requirements.

2. **Ad hoc queries**
    - MongoDB supports searching by field, range queries, and regular expression searches. **Queries** can be made to return specific fields within documents.

3. **Replication**
    - MongoDB can provide **high availability** with replica sets.
    - A replica set consists of two or more mongoDB instances.
    - Each replica set member may act in the role of the primary or secondary replica at any time.
    - The primary replica is the main server which interacts with the client and performs all the read/write operations.

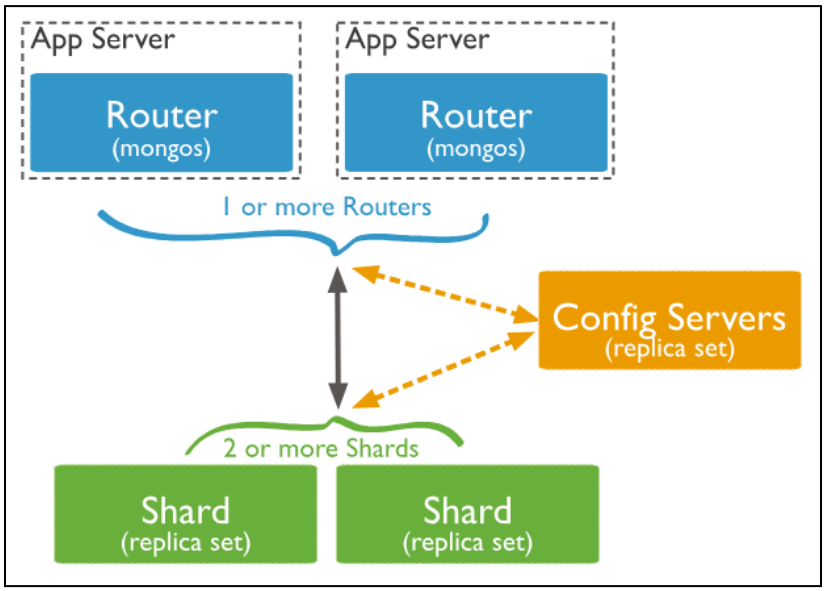
- The Secondary replicas maintain a copy of the data of the primary using built-in replication.
- When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.

4. **Indexing**
- Indexes can be created to improve the performance of searches within MongoDB.
- Any field in a MongoDB document can be indexed.

5. **Load balancing**
- MongoDB uses the concept of **sharding** to scale horizontally by splitting data across multiple MongoDB instances.
- MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

6. **Sharding** (next question)

Q.3 Explain Sharding



- Supports Sharding which is the process of dividing large datasets across multiple collections to ensure that queries can be executed efficiently.
- Sharding helps in horizontal scaling by partitioning the large collection (dataset) into smaller discrete parts called shards.
- Splitting up large Collections into Shards allows MongoDB to execute queries without putting much load on the Server.
- MongoDB Sharding can be implemented by creating a Cluster of MongoDB Instances.

GFG:
- Sharding is a method for distributing large collection(dataset) and allocating it across multiple servers. MongoDB uses sharding to help deployment with very big data sets and high volume operations.
- Sharding combines more devices to carry data extension and the needs of read and write

operations.
- Sharding determines the problem with horizontal scaling. It breaks the system dataset and store it over multiple servers, adding new servers to increase the volume as needed.
- Now, instead of one signal as primary, we have multiple servers called Shard.

Advantages of Sharding

- Sharding adds more server to a data field automatically adjust data loads across various servers.
- The number of operations each shard manage got reduced.
- It also increases the write capacity by splitting the write load over multiple instances.
- It gives high availability due to the deployment of replica servers for shard and config.
- Total capacity will get increased by adding multiple shards.

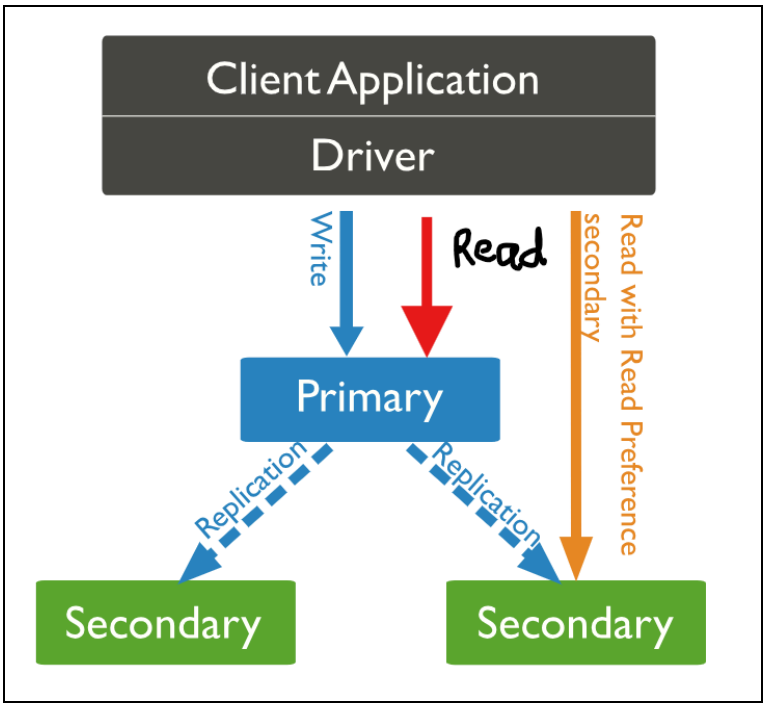
[Sharding — MongoDB Manual](#) (extra)

Q.4 Explain Replication

In MongoDB, replication refers to the process of synchronizing data across multiple servers to ensure data availability, fault tolerance, and scalability. Replication involves maintaining multiple copies of the same data across different servers, known as replica sets.

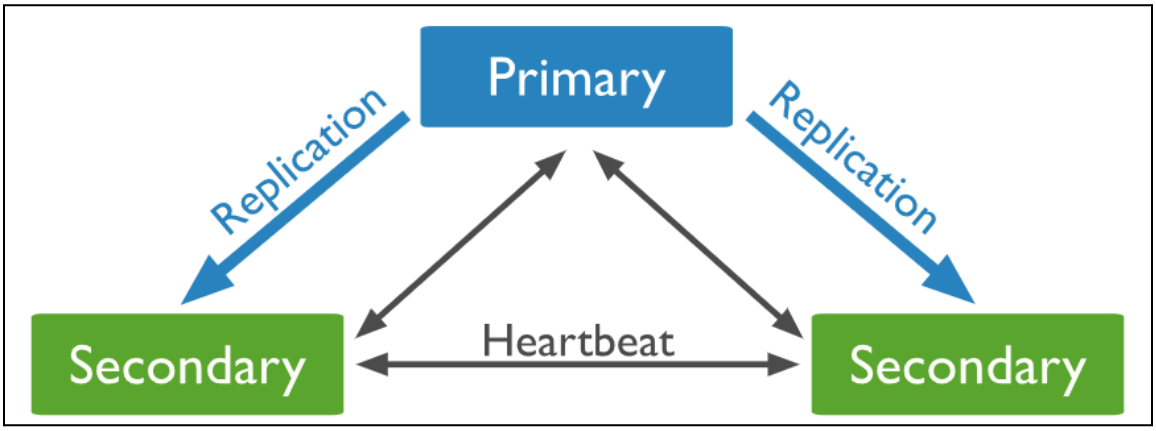
Here's how replication works in MongoDB:

- Replica Set: A replica set is a group of MongoDB servers that maintain the same data set. Within a replica set, there are one primary node and multiple secondary nodes. The primary node is responsible for receiving all write operations and replicating the data changes to the secondary nodes.
- Primary Node: The primary node is the only node that can accept write operations. All write operations, such as inserts, updates, and deletes, are first processed by the primary node. After processing, the changes are replicated to the secondary nodes.
- Secondary Nodes: Secondary nodes replicate data from the primary node. They can also serve read operations, which can help distribute the read workload and improve read performance. However, secondary nodes cannot accept write operations.



``

- The secondaries replicate the primary's oplog and apply the operations to their data sets such that the secondaries' data sets reflect the primary's data set. If the primary is unavailable, an eligible secondary will hold an election to elect itself the new primary.



Q.5 Discuss RDBMS v/s MongoDB

RDBMS	MongoDB
It is a relational database.	It is a non-relational and document-oriented database.

Not suitable for hierarchical data storage.	Suitable for hierarchical data storage.
It is vertically scalable i.e increasing RAM.	It is horizontally scalable i.e we can add more servers.
It has a predefined schema.	It has a dynamic schema.
It is quite vulnerable to SQL injection.	It is not affected by SQL injection.
It centers around ACID properties (Atomicity, Consistency, Isolation, and Durability).	It centers around the CAP theorem (Consistency, Availability, and Partition tolerance).
It is row-based.	It is document-based.
It is slower in comparison with MongoDB.	It is almost 100 times faster than RDBMS.
Supports complex joins.	No support for complex joins.
It is column-based.	It is field-based.
It does not provide JavaScript client for querying.	It provides a JavaScript client for querying.

It supports SQL query language only.	It supports JSON query language along with SQL.

Q.6 Discuss Administering User Accounts and User Roles

Administering user accounts and user roles in MongoDB is crucial for maintaining security and controlling access to databases and collections. MongoDB provides a flexible and robust system for managing users and their permissions. Here's a discussion on administering user accounts and user roles in MongoDB:

- Authentication: MongoDB supports various authentication mechanisms such as SCRAM (Salted Challenge Response Authentication Mechanism) and LDAP (Lightweight Directory Access Protocol). Authentication ensures that only authorized users can access the database.

- User Account Creation: To create a user account in MongoDB, you can use the db.createUser() method. This method allows you to specify the username, password, and roles for the user.
- For example:
  ```
  db.createUser({
    user: "myUser",
    pwd: "myPassword",
    roles: [ { role: "readWrite", db: "myDatabase" } ]
  })
  ```

- User Roles: MongoDB provides several built-in roles that define the privileges a user has within a database or collection. Some common built-in roles include:
  - read: Provides read-only access to a database or collection.
  - readWrite: Provides read and write access to a database or collection.
  - dbAdmin: Provides administrative privileges for a specific database.
  - userAdmin: Provides privileges to create and manage users in a specific database.
  - clusterAdmin: Provides administrative privileges for the entire cluster.
- Custom Roles: MongoDB allows you to define custom roles with specific sets of privileges tailored to your application's requirements. Custom roles can be created using the db.createRole() method and then assigned to users. Custom roles can have permissions at the database level or collection level.

- Assigning Roles to Users: Once roles are defined, you can assign them to users using the db.grantRolesToUser() method.
- For example:
  `db.grantRolesToUser("myUser", [ { role: "read", db: "myDatabase" } ])`
- Revoking Roles: Similarly, you can revoke roles from users using the db.revokeRolesFromUser() method.

- Auditing: MongoDB Enterprise Edition provides auditing capabilities that allow you to track user actions, authentication attempts, and other important events for security and compliance

purposes.

- Encryption: MongoDB supports encryption at rest and in transit to ensure data security. You can enable encryption for data stored on disk and encrypt communication between MongoDB clients and servers using TLS/SSL.

Q.7 Explain User Action Privileges

In MongoDB, user action privileges define the specific actions that a user is allowed to perform on a particular resource within the database. These privileges play a crucial role in controlling access to data and ensuring the security and integrity of the database. Let's delve deeper into the components and workings of user action privileges in MongoDB:

1. **Resource:**
   - The resource component of a privilege specifies the target or scope of the action. It can include:
     - Databases: Such as "sales", "inventory", etc.
     - Collections: Like "customers", "orders", etc.
     - Cluster: Referring to the entire MongoDB cluster.
2. **Action:**
   - The action component describes the specific type of behavior permitted by the privilege. Common actions include:
     - Read: Allows users to view data.
     - Write: Permits users to insert, update, or delete data.
     - Update: Allows users to modify existing data.
     - Delete: Grants users the ability to remove data.
     - Administrative Actions: Such as creating or dropping collections, indexes, users, etc.
3. **Privilege Assignment:**
   - Users receive privileges through role assignments. A role is a set of privileges that are grouped together for convenience.
   - A user can be assigned multiple roles, each granting different sets of privileges.
   - Roles can be predefined (built-in) or custom-defined, depending on the specific needs of the application.

   Example:
   - Let's consider a scenario where we have a MongoDB database for a retail application.
   - We might define roles such as "sales_manager" and "inventory_clerk".
   - The "sales_manager" role might include privileges to read and write data in the "sales" collection, as well as administrative privileges to manage users.
   - Meanwhile, the "inventory_clerk" role might only have read privileges on the "inventory" collection.
   - Users are then assigned these roles based on their responsibilities within the organization.
4. **Enabling Access Control:**
   - MongoDB's access control mechanism ensures that users need to authenticate themselves before interacting with the database.
   - Once access control is enabled, users must provide valid credentials (username and password) to access the database, ensuring that only authorized users can perform actions.

User action privileges in MongoDB define what actions users are permitted to perform on specific resources within the database. By carefully managing these privileges through role assignments, MongoDB administrators can control access to data and ensure the security and integrity of the database system.

Q.8 Discuss Configuring Access Control with example.

Configuring access control in MongoDB involves enabling authentication, creating users, defining roles, and assigning privileges. Let's go through the steps with an example:

1. **Enable Authentication:**
   - MongoDB's access control is disabled by default. To enable it, you need to start the MongoDB server with the --auth option or set security.authorization to enabled in the configuration file (mongod.conf).
   - For example, in the mongod.conf file

     security:
     authorization: enabled

2**. Create Administrative User:**

- Before configuring other users and roles, it's best practice to create an administrative user who can manage the database.
- Connect to the MongoDB instance as a user with administrative privileges (e.g., the default "admin" database):

use admin

Create an administrative user with roles that allow user management:

```
db.createUser({
  user: "adminUser",
  pwd: "adminPassword",
  roles: ["userAdminAnyDatabase", "dbAdminAnyDatabase", "clusterAdmin"]
})
```

3. Define Custom Roles (Optional):

- Depending on your application's requirements, you may need to define custom roles.
- For example, if your application involves managing customer data, you might create a role with read and write privileges on the "customers" collection.
- Custom roles can be created using the db.createRole() method or the db.grantRolesToUser() method.

4. Create Application Users:

- Create users for your application with appropriate roles and privileges.
- Connect to the MongoDB instance as an administrative user.

use admin

Create a user for your application database with specific roles:

```
use myAppDatabase
db.createUser({
  user: "appUser",
```

pwd: "appPassword",
roles: ["readWrite"]
})

5. Test Access Control:

- Restart the MongoDB instance with authentication enabled.
- Connect to the MongoDB instance using the created application user credentials:

mongo -u appUser -p appPassword --authenticationDatabase myAppDatabase

By following these steps, you can configure access control in MongoDB, ensuring that only authorized users can access and manipulate data within the database. It's essential to regularly review and update user privileges to maintain the security and integrity of your MongoDB deployment.

Q.9 Explain Accessing and Manipulating Databases commands.

https://www.javatpoint.com/mongodb-user-management-methods

Q.10 Explain examining the rules of REST APIs.

- REST, which stands for Representational State Transfer, is a set of architectural principles used for designing networked applications. APIs that adhere to these principles are known as RESTful APIs. RESTful APIs use HTTP requests to manage data and interact with services over the web. They are stateless, meaning that each request from a client to the server must contain all the information the server needs to fulfill the request, without relying on any stored context on the server.
- Here are some of the core rules and principles associated with RESTful API design:

1. Client-Server Architecture
   - RESTful APIs follow a client-server architecture where the client and server act independently. The client is responsible for the user interface and user-related state, while the server manages the data and state management. This separation allows both components to evolve independently, enhancing the scalability and portability of the application.
2. Statelessness
   - In REST, the server does not store any state about the client session on the server side. Each request from the client to the server must contain all the necessary information to understand and complete the request. This means that session state is kept entirely on the client.

3. Cacheability
    - Responses from RESTful APIs should explicitly state whether they are cacheable or not. If responses are cacheable, the client can reuse the response data for equivalent requests in the future. This can significantly reduce the interaction between client and server, improving efficiency and scalability.
4. Uniform Interface
    - One of the key constraints of REST is the uniform interface between clients and servers. This simplifies and decouples the architecture, allowing each part to evolve independently. The uniform interface is characterized by four guiding principles:
    - Resource-Based: Individual resources are identified in requests using URIs as resource identifiers.
    - Manipulation of Resources Through Representations: When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource on the server, provided it has permission to do so.
    - Self-descriptive Messages: Each message includes enough information to describe how to process the message. For instance, which parser to invoke can be decided based on the media type (Content-Type header).
    - Hypermedia as the Engine of Application State (HATEOAS): Clients deliver state via body contents, query-string parameters, request headers, and the requested URI (the resource name). Services deliver state to clients via body content, response codes, and response headers. This is similar to a web browser accessing web pages by following links and using URIs.
5. Layered System
    - A client cannot ordinarily tell whether it is connected directly to the end server or to an intermediary along the way. Intermediate servers improve system scalability by enabling load balancing and by providing shared caches. Layers can also enforce security policies.
6. Code on Demand (Optional)
    - Servers can temporarily extend or customize the functionality of a client by transferring executable code. This is the only optional constraint of REST and can be used, for example, to send client-side scripts like JavaScript in web applications.

    - Example of a RESTful API Call:
      GET Request: Fetch data from a specified resource.
      GET /users/12345

- This GET method is used to request data about a user from a server using the user's ID (12345). The response should be a representation of the user, typically in JSON or XML format.
- Adhering to these REST principles ensures that an API is scalable, flexible, simple to use, and performant. Systems designed around REST APIs are easier to extend over time and can interact with a broad range of clients due to their standard, web-friendly nature.

Q.11 What are the advantages of MongoDB?

- Document-oriented
    - Since MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very **flexible and adaptable** to real business

world situation and requirements.

- Ad hoc queries
  - MongoDB supports searching by field, range queries, and regular expression searches. **Queries** can be made to return specific fields within documents.
- Replication
  - MongoDB can provide **high availability** with replica sets.
  - A replica set consists of two or more mongoDB instances.
  - Each replica set member may act in the role of the primary or secondary replica at any time.
  - The primary replica is the main server which interacts with the client and performs all the read/write operations.
  - The Secondary replicas maintain a copy of the data of the primary using built-in replication.
  - When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.
- Indexing
  - Indexes can be created to improve the performance of searches within MongoDB.
  - Any field in a MongoDB document can be indexed.
- Load balancing
  - MongoDB uses the concept of **sharding** to scale horizontally by splitting data across multiple MongoDB instances.
  - MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.
- Sharding
  - Supports Sharding which is the process of dividing large datasets across multiple collections to ensure that queries can be executed efficiently.
  - Splitting up large Collections into Shards allows MongoDB to execute queries without putting much load on the Server.
  - MongoDB Sharding can be implemented by creating a Cluster of MongoDB Instances.

Q.12 What is a Document in MongoDB?

- A record in a MongoDB collection is basically called a document.
- The document, in turn, will consist of field name and values.
- In MongoDB, a document is the basic unit of data, similar to a row in relational databases. However, MongoDB is a NoSQL database designed to store data in a flexible, JSON-like format called BSON (Binary JSON). Each document in MongoDB is a data structure composed of field and value pairs.
- Fields are similar to columns in a SQL database, and values can include various data types such as strings, numbers, arrays, and even other documents.
- This structure allows documents to have different fields and data types from one another, which provides a high level of flexibility compared to the fixed schema of a traditional relational database.
- Flexible Schema: Documents in the same collection (analogous to a table in a relational database) do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.
- Document Format: Documents are essentially JSON-like field/value pairs. BSON, MongoDB's serialization format, extends JSON with additional data types such as Date and binary data.
- Document ID: Each document stored in a MongoDB collection has a unique identifier (_id) that serves as a

primary key. The _id field is automatically added by MongoDB to each document unless specified by the user.

- Nested Structures: Documents can contain nested documents and arrays, allowing complex data structures to be represented directly within a single document.

Eg:

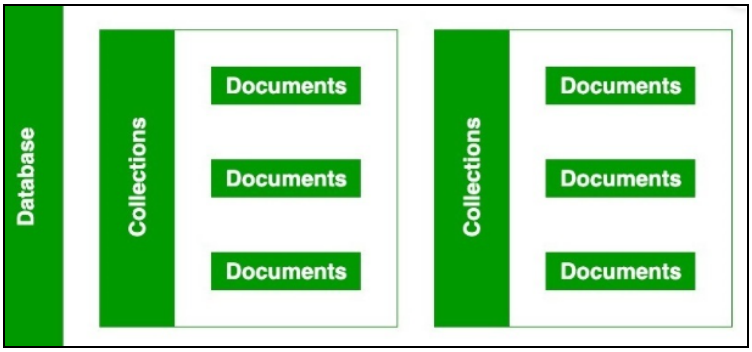
```
{
  "_id": ObjectId("507f191e810c19729de860ea"),
  "name": "John Doe",
  "age": 30,
  "address": {
    "street": "123 Elm St",
    "city": "Somewhere",
    "state": "CA"
  },
  "hobbies": ["reading", "games", "hiking"]
}
```

Q.13 What is a Collection in MongoDB?

- This is a grouping of MongoDB documents.
- A collection is the equivalent of a table that is created in any other RDMS such as Oracle or MS
  SQL.
- A collection exists within a single database.
- The collection is not Schema-dependent.
- Documents with different numbers and types of fields can be inserted.
- Create Collection (new or switches): use <collection_name>
- Show collections in the current DB: show <collection_name>

Q.14 Explain the working of MongoDB in detail.

- MongoDB environment gives you a server that you can start and then create multiple databases using MongoDB.
- Because of its NoSQL database, the data is stored in collections and documents. Hence the database, collection, and documents are related to each other as shown below:



- The MongoDB database contains collections just like the MYSQL database contains tables. You are allowed to create multiple databases and multiple collections.
- Now inside of the collection we have documents. These documents contain the data we want to store in the MongoDB database and a single collection can contain multiple documents you are schema-less means one document doesn't need to be similar to another.
- The documents are created using the fields. Fields are key-value pairs in the documents, it is just like columns in the relation database. The value of the fields can be of any BSON data type like double, string, boolean, etc.
- The data stored in the MongoDB is in the format of BSON documents. Here, BSON stands for Binary representation of JSON documents. In other words, in the backend, the MongoDB server converts the JSON data into a binary form that is known as BSON, and this BSON is stored and queried more efficiently.
- In MongoDB documents, you are allowed to store nested data. This nesting of data allows you to create complex relations between data and store them in the same document which makes the working and fetching of data extremely efficient as compared to SQL. In SQL, you need to write complex joins to get the data from table 1 and table 2. The maximum size of the BSON document is 16MB.

Q.15 List out applications of MongoDB.

1. **Web Applications:**
   MongoDB serves as the primary data store for many web applications. The popular MEAN stack (MongoDB, ExpressJS, AngularJS, and NodeJS) employs MongoDB as its backend data store.
   It’s well-suited for applications that require fast and scalable data storage.
2. **Big Data Systems:**
   MongoDB works well with unstructured data, making it ideal for Big Data systems and MapReduce applications.
   It’s commonly used in scenarios where data doesn’t fit neatly into traditional relational tables.
3. **Social Networking Applications:**
   MongoDB’s schema-less nature allows for flexible data modeling, making it a good fit for social networking platforms.
   It’s used by platforms like Facebook, where billions of users generate and access data constantly.

4. **Content Management Systems (CMS):**
   MongoDB's ability to handle large volumes of data efficiently makes it suitable for content management systems.
   It simplifies content storage and retrieval for websites and applications.
5. **Real-Time Analytics:**
   MongoDB's fast read and write capabilities are advantageous for real-time analytics.
   It's used in scenarios where quick data access is critical.
6. **Mobile Applications:**
   Mobile apps often require seamless data synchronization between devices and servers.
   MongoDB's flexibility and scalability make it a good choice for mobile app backends.
7. **Internet of Things (IoT):**
   MongoDB can handle diverse data types, making it useful for IoT applications.
   It stores sensor data, device logs, and other IoT-related information.
8. **Energy Utilities and Telecommunications:**
   Energy utilities and telecom companies employ MongoDB to manage large datasets efficiently.
   It helps track usage, monitor infrastructure, and analyze customer data.
9. **Healthcare and Medical Applications:**
   Healthcare systems benefit from MongoDB's ability to handle complex patient records, medical images, and other health-related data.
10. **Government Agencies:**
    Even government agencies use MongoDB to store and manage vast amounts of data.

Q.16 Explain MongoDB CRUD Operations with an example.

CRUD operations describe the conventions of a user-interface that let users view, search, and modify parts of the database.

MongoDB documents are modified by connecting to a server, querying the proper documents,
and then changing the setting properties before sending the data back to the database to be updated. CRUD is data-oriented, and it's standardized according to HTTP action verbs.

When it comes to the individual CRUD operations:

. The Create operation is used to insert new documents in the MongoDB database.
. The Read operation is used to query a document in the database.
. The Update operation is used to modify existing documents in the database.
. The Delete operation is used to remove documents in the database.

a) **Create Operations**

For MongoDB CRUD, if the specified collection doesn't exist, the create operation will create the collection when it's executed.

MongoDB provides two different create operations that you can use to insert documents into a collection:

. db.collection.insertOne()
. db.collection.insertMany()

insertOne()

As the namesake, insertOne() allows you to insert one document into the collection.

```
db.student. insertOne({
name: "Marsh",
age:6
})
```

b) Read Operations

The read operations allow you to supply special query filters and criteria that let you specify which documents you want.

MongoDB has two methods of reading documents from a collection:

. db.collection.find()
. db.collection.findOne()

find()

In order to get all the documents from a collection, we can simply use the find() method on our chosen collection.

db. students. find()

{ "_id" : ObjectId("5fd98ea9ce6e8850d88270b5"), "name" : "Marsh", "age" : 4

c) Update Operations

Like create operations, update operations operate on a single collection, and they are atomic at a single document level. An update operation takes filters and criteria to select the documents you want to update.

You should be careful when updating documents, as updates are permanent and can't be rolled back. This applies to delete operations as well.

For MongoDB CRUD, there are three different methods of updating documents:

. db.collection.updateOne()
. db.collection.updateMany()
. db.collection.replaceOne()

updateOne()

We can update a currently existing record and change a single document with an update operation. To do this, we use the updateOne() method on a chosen collection, which here is "RecordsDB." To update a document, we provide the method with two arguments: an update filter and an update action.

db.students.updateOne({name: "Marsh"}, {$set: {age:22}})

d) Delete Operations

Delete operations operate on a single collection, like update and create operations. Delete operations are also atomic for a single document.

MongoDB has two different methods of deleting records from a collection:

. db.collection.deleteOne()
. db.collection.deleteMany()

deleteOne()

deleteOne() is used to remove a document from a specified collection on the MongoDB server. A filter criteria is used to specify the item to delete. It deletes the first record that matches the provided filter.

db.RecordsDB.deleteOne({name:"Marsh"})

Q.17 Write a short note on Mongoose schemas and Models.

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a straightforward way to define schemas for your MongoDB documents and create models that are

based on these schemas.

a) **Schema**:
A Mongoose schema defines the structure of documents within a MongoDB collection. It specifies the shape of the documents, including the fields, their types, and any additional properties.

Each field in a schema is defined by a key-value pair, where the key is the name of the field and the value is an object that specifies the field's type and other properties. For example, { name: { type: String, required: true } } defines a name field that must be a string and is required.

Mongoose supports various schema types, such as String, Number, Date, Boolean, ObjectId, etc., allowing you to define complex data structures.

Schemas can also define validation rules for each field, ensuring that data inserted into the database meets certain criteria. For example, { age: { type: Number, min: 18 } } specifies that the age field must be a number greater than or equal to 18.

You can specify default values for fields, which are used when a document is created without providing a value for that field.

```
const userSchema = new mongoose.Schema({
        name: {
                type: String,
                required: true
        },
        email: {
                type: String,
                required: true,
                unique: true
        },
        age: {
                type: Number,
                default: 18
        }
});
```

b) **Model**:
A Mongoose model is a constructor function that creates instances of documents based on a schema. Each document represents a record in a MongoDB collection.

Models provide methods for performing CRUD operations on documents, such as create, find, findOne, update, and delete.

Models support middleware functions that allow you to execute logic before or after certain operations, such as save, validate, update, etc.

Static and Instance Methods: You can define static and instance methods on models to encapsulate reusable logic for working with documents.

Models support query building, allowing you to construct complex queries using chaining methods like where, sort, limit, and skip.

const User = mongoose.model('User', userSchema);

You can then use the User model to create new user documents, find existing ones, update them, and delete them.

Q.18 Explain REST API in detail.

Representational State Transfer (REST) is an architectural style that defines a set of constraints to be used for creating web services. REST API is a way of accessing web services in a simple and flexible way without having any processing.

Working
A request is sent from client to server in the form of a web URL as HTTP GET or POST or PUT or DELETE request. After that, a response comes back from the server in the form of a resource which can be anything like HTML, XML, Image, or JSON. But now JSON is the most popular format being used in Web Services.

In HTTP there are five methods that are commonly used in a REST-based Architecture i.e., POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations respectively.

1) GET: The HTTP GET method is used to read (or retrieve) a representation of a resource. In the safe path, GET returns a representation in XML or JSON and an HTTP response code of 200 (OK). In an error case, it most often returns a 404 (NOT FOUND) or 400 (BAD REQUEST).

   Request :
   GET:/api/students

2) POST: The POST verb is most often utilized to create new resources. On successful creation,

return HTTP status 201, returning a Location header with a link to the newly-created resource with the 201 HTTP status.

Request:
POST:/api/students
Body:
{"name":"Raj"}

3) PUT: It is used for updating the capabilities.On successful update, return 200 (or 204 if not returning any content in the body) from a PUT.

Request:
PUT or PATCH:/api/students/1
Body:
{"name":"Shubham"}

4) PATCH: It is used to modify capabilities. The PATCH request only needs to contain the changes to the resource, not the complete resource.

5) DELETE: It is used to delete a resource identified by a URI. On successful deletion, return HTTP status 200 (OK) along with a response body.

Request:
DELETE:/api/students/1

Features of REST API

- Resource-Based: REST APIs are built around resources, which are any kind of object, data, or service that can be accessed, created, or manipulated. Resources are identified by URIs.
- Uniform Interface: REST APIs use a uniform interface, typically HTTP, with standard methods (GET, POST, PUT, DELETE) to perform actions on resources. This simplifies the architecture and makes the API easy to use and understand.
- Stateless: REST APIs are stateless, meaning that each request from a client to the server must contain all the information necessary to understand and process the request. The server does not store any client state between requests.
- Client-Server Architecture: REST APIs follow a client-server architecture, where the client and server are separate and communicate over a network using standard protocols.

Q.19 Enlist different types of HTTP requests supported by REST API. (source: phind.com)

REST APIs support a variety of HTTP request methods, each designed for a specific purpose. The most commonly used HTTP request methods in RESTful web

development are:

GET: Used to retrieve data from a server. It is a safe and idempotent method, meaning it can be called multiple times without changing the server's state.

POST: Sends data to the server for processing, such as submitting form data or uploading a file. It is not idempotent, meaning calling it multiple times can result in different outcomes.

PUT: Replaces the entire content of a resource on the server with the data provided in the request. It is idempotent, meaning calling it multiple times with the same data will have the same effect as calling it once.

PATCH: Partially updates a resource on the server with the data provided in the request. Unlike PUT, it only modifies the parts of the resource specified in the request.

DELETE: Removes a resource from the server. It is idempotent, meaning calling it multiple times will have the same effect as calling it once.

Additionally, there are other HTTP methods like OPTIONS, which is used to describe the communication options for the target resource, and HEAD, which is similar to GET but only retrieves the headers of the response.

Q.20 List out HTTP response codes returned by REST API. (source: phind.com)

REST APIs use HTTP response codes to indicate the outcome of a request. These codes are divided into five categories, each providing information about the success, redirection, client error, or server error of the request. Here are some of the most common HTTP response codes you might encounter:

1xx: Informational

1) 100 Continue: The server has received the request headers, and the client should proceed to send the request body.
2) 101 Switching Protocols: The requester has asked the server to switch protocols, and the server has agreed to do so.

2xx: Success

1) 200 OK: The request has succeeded. The information returned with the response is dependent on the method used in the request.
2) 201 Created: The request has been fulfilled, and a new resource was created as a result.
3) 202 Accepted: The request has been accepted for processing, but the processing has not been completed.
4) 204 No Content: The server successfully processed the request and is not returning any content.

3xx: Redirection

1) 301 Moved Permanently: The URL of the requested resource has been changed permanently. The new URL is given in the response.
2) 302 Found: This response code means that the URI of requested resource has been temporarily changed.
3) 304 Not Modified: Indicates that the resource has not been modified since the last request.

4xx: Client Error

1) 400 Bad Request: The server cannot or will not process the request due to something that is perceived to be a client error.
2) 401 Unauthorized: The request has not been applied because it lacks valid authentication credentials for the target resource.
3) 403 Forbidden: The server understood the request, but it refuses to authorize it.
4) 404 Not Found: The server can not find the requested resource.

5xx: Server Error

1) 500 Internal Server Error: The server encountered an unexpected condition that prevented it from fulfilling the request.
2) 502 Bad Gateway: The server, while acting as a gateway or proxy, received an invalid response from the upstream server.
3) 503 Service Unavailable: The server is currently unable to handle the request due to a temporary overload or scheduled maintenance.

Q.21 Explain API design patterns(GPT)

API design patterns are reusable solutions to common problems encountered when designing APIs (Application Programming Interfaces). These patterns provide guidance on structuring endpoints, handling data, managing resources, and ensuring reliability and scalability. Here are some common API design patterns:

1) **RESTful APIs:** Representational State Transfer (REST) is a popular architectural style for designing networked applications. RESTful APIs use standard HTTP methods (GET, POST, PUT, DELETE) to perform CRUD (Create, Read, Update, Delete) operations on resources identified by URLs. They follow the principles of statelessness, uniform interface, and resource-based interactions.
2) **GraphQL:** Unlike REST, GraphQL allows clients to query exactly the data they need, reducing over-fetching and under-fetching of data. It provides a single

endpoint for querying and manipulating data, allowing clients to define the structure of the response.

3) **RPC (Remote Procedure Call):** RPC APIs allow clients to invoke procedures or methods on remote servers as if they were local. Common implementations include gRPC, which uses Protocol Buffers for efficient serialization, and JSON-RPC, which uses JSON for data interchange.
4) **Observer Pattern:** This pattern allows clients to subscribe to changes in resources and receive notifications when those changes occur. It's useful for real-time updates and event-driven architectures.
5) **Pagination:** When dealing with large datasets, pagination allows clients to retrieve a subset of the data in manageable chunks. Pagination parameters, such as page number and page size, are typically included in API requests.
6) **Rate Limiting:** Rate limiting restricts the number of requests a client can make to an API within a specified time frame. This helps prevent abuse, ensures fair usage of resources, and maintains API performance and availability.
7) **Versioning:** APIs evolve over time, and versioning allows for backward compatibility while introducing changes. Versioning can be done through URL paths (e.g., /v1/resource), custom headers (e.g., Accept-Version), or query parameters (e.g., ?version=1).
8) **Stateless Authentication:** Stateless authentication mechanisms, such as JWT (JSON Web Tokens), allow clients to authenticate requests without the need for server-side sessions. Tokens are signed and contain user information, enabling stateless and scalable authentication.
9) **Circuit Breaker:** The circuit breaker pattern prevents cascading failures by temporarily halting requests to a service that is experiencing issues. It monitors the health of downstream services and opens the circuit when failures exceed a threshold, redirecting requests to a fallback mechanism.
10) **Idempotent Operations:** Idempotent operations have the same effect whether executed once or multiple times, making them safe to retry in case of network failures or timeouts. HTTP methods like GET, PUT, and DELETE are typically idempotent, while POST is not.

Module 5: Flask

Q.1 Explain App Routing

In Flask, app routing refers to the process of mapping URLs to view functions within your application. When a user requests a particular URL, Flask determines which Python function should handle the request and generate the appropriate response. This mapping is established using the **@app.route()** decorator.

a) ***Code:***

```
@app.route('/home')
def home():
   return "hello, welcome to our website";
```

We defined routes using the @app.route() decorator. This route ('/home') corresponds to the homepage

Flask facilitates us to add the variable part to the URL by using the section. We can reuse the variable by adding that as a parameter into the view function.

b) ***Code:***

```
@app.route('/home/<name>')
def home(name):
   return "hello,"+name;
```

Now on browser if we type https://localhost:5000/home/Shubham

It will display

hello Shubham

There is one more approach to perform routing for the flask web application that can be done by using the add_url() function of the Flask class

add_url_rule(<url rule>, <endpoint>, <view function>)

This function is mainly used in the case if the view function is not given and we need to connect a view function to an endpoint externally by using this function.

c) ***Code:***

```
def about():
   return "This is about page";

app.add_url_rule("/about","about",about)
```

Q.2 Discuss URL Building.

5.4 FLASK URL BUILDING

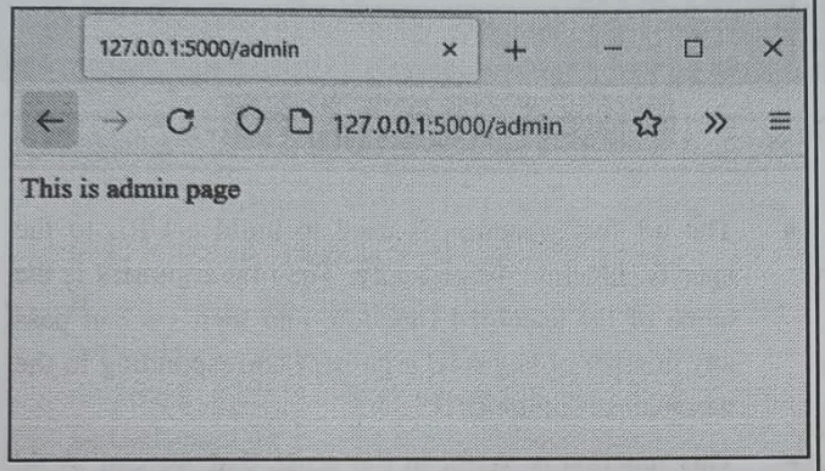
- The url_for() function is used to build a URL to the specific function dynamically. The first argument is the name of the specified function, and then we can pass any number of keyword argument corresponding to the variable part of the URL.
- This function is useful in the sense that we can avoid hard-coding the URLs into the templates by dynamically building them using this function.

Example

```
from flask import *
app = Flask(__name__)
@app.route('/admin')
def admin():
    return 'This is admin page'
@app.route('/librarian')
def librarian():
    return 'This is librarian page'
@app.route('/student')
def student():
    return 'This is student page'
@app.route('/user/<name>')
def user(name):
    if name == 'admin':
        return redirect(url_for('admin'))
    if name == 'librarian':
        return redirect(url_for('librarian'))
    if name == 'student':
        return redirect(url_for('student'))
if __name__ =='__main__':
    app.run(debug = True)
```

- The above example describes the library management system which can be used by the three types of users, i.e., admin, librarian, and student.
- There is a specific function named user() which recognizes the user the redirect the user to the exact function which contains the implementation for this particular function.

Output



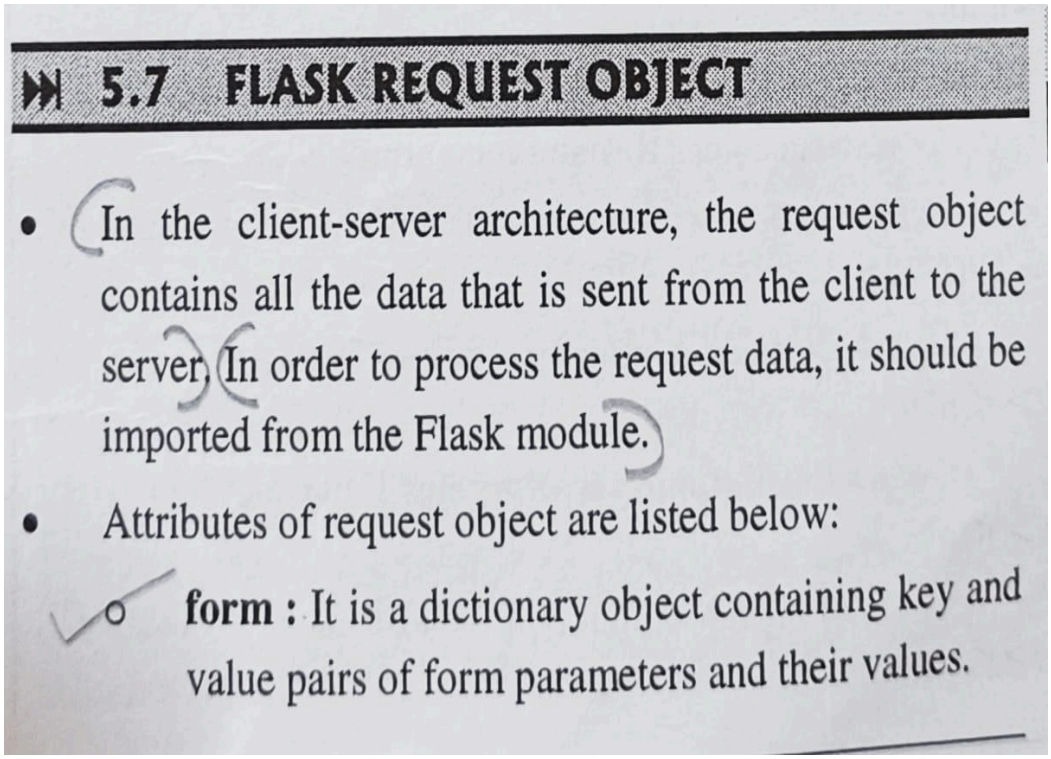
- For example, the URL http://localhost:5000/user/admin is redirected to the URL http://localhost:5000/admin, the URL localhost:5000/user/librarian, is redirected to the URL http://localhost:5000/librarian, the URL http://localhost:5000/user/student is redirected to the URL http://localhost/student.

☞ Benefits of the Dynamic URL Building

1. It avoids hard coding of the URLs.
2. We can change the URLs dynamically instead of remembering the manually changed hard-coded URLs.
3. URL building handles the escaping of special characters and Unicode data transparently.
4. The generated paths are always absolute, avoiding unexpected behavior of relative paths in browsers.
5. If your application is placed outside the URL root, for example, in /myapplication instead of /, url_for() properly handles that for you.

Q.3 Explain Flask HTTP Methods

Q.4 Discuss Flask Request Object



5.7 FLASK REQUEST OBJECT

- In the client-server architecture, the request object contains all the data that is sent from the client to the server. In order to process the request data, it should be imported from the Flask module.
- Attributes of request object are listed below:
  - **form :** It is a dictionary object containing key and value pairs of form parameters and their values.

- **args :** It is parsed contents of query string which is part of URL after question mark (?).
- **cookies :** It is the dictionary object holding Cookie names and values.
- **files :** It contains the data pertaining to uploaded file.
- **method :** It is the current request method (get or post).

- **Example :** In the following example, the / URL renders a web page customer.html that contains a form which is used to take the customer details as the input from the customer. The data filled in this form is posted to the /success URL which triggers the print_data() function. The print_data() function collects all the data from the request object and renders the result_data.html file which shows all the data on the web page.

script.py

```
from flask import *
app = Flask(__name__)

@app.route('/')
def customer():
  return render_template('customer.html')

@app.route('/success',methods = ['POST', 'GET'])
def print_data():
  if request.method == 'POST':
      result = request.form
      return render_template("result_data.html",result =
result)

if __name__ == '__main__':
  app.run(debug = True)
```

customer.html

```
<html>
  <body>
      <h3>Customer Registration Form</h3>
      <form action = "http://localhost:5000/success" method
= "POST">
          <p>Name <input type = "text" name = "name"
/></p>
        <p>Email <input type = "email" name = "email"
/></p>
        <p>Contact <input type = "text" name = "contact"
/></p>
```

```
        <p>Pin code <input type ="text" name = "pin"
/></p>
        <p><input type = "submit" value = "submit"
/></p>
      </form>
  </body>
</html>
result_data.html
<!doctype html>
<html>
  <body>
      <p><strong>Thanks for the
registration.</strong></p>
      <table border = 1>
        {% for key, value in result.items() %}
          <tr>
            <th> {{ key }} </th>
            <td> {{ value }} </td>
          </tr>
        {% endfor %}
      </table>
  </body>
</html>
```

Q.5 Explain Flask cookies with an example

5.8 FLASK COOKIES

- The cookies are stored in the form of text files on the client's machine. Cookies are used to track the user's activities on the web and reflect some suggestions according to the user's choices to enhance the user's experience.
- Cookies are set by the server on the client's machine which will be associated with the client's request to that particular server in all future transactions until the lifetime of the cookie expires or it is deleted by the specific web page on the server.

Setting a Cookie:
In Flask, we use set_cookie() method of the response object to set cookies. The syntax of set_cookie() method is as follows:

response.set_cookie('cookie_name', 'cookie_value',
max_age=expiration_time_in_seconds)

Accessing a Cookie:
To access the cookie, we use the cookie attribute of the request object. The cookie attribute is a dictionary like attribute which contains all the cookies sent by the browser.

Syntax:
cookie_value = request.cookies.get('cookie_name')

Deleting a cookie:
To delete a cookie call set_cookie() method withe name of the cookie and any value and set the max_age argument to 0.

Syntax:
response.set_cookie('cookie_name', '', max_age=0)

Example:

```
from flask import Flask, request, make_response, jsonify

app = Flask(__name__)

# Route to set a cookie
@app.route('/set_cookie')
def set_cookie():
    response = make_response(jsonify({'message': 'Cookie set successfully'}))
    response.set_cookie('username', 'John', max_age=3600)  # Max age in
seconds (here, 1 hour)
    return response

# Route to get the value of a cookie
@app.route('/get_cookie')
def get_cookie():
    username = request.cookies.get('username')
```

```
    if username:
        return jsonify({'username': username})
    else:
        return jsonify({'message': 'Cookie not found'})

# Route to delete a cookie
@app.route('/delete_cookie')
def delete_cookie():
    response = make_response(jsonify({'message': 'Cookie deleted
successfully'}))
    response.set_cookie('username', '', max_age=0)  # Setting expiration in the
past to delete cookie
    return response

if __name__ == '__main__':
    app.run(debug=True)
```

Q.6 Explain File Uploading in Flask

5.9 FLASK FILE UPLOADING

- File uploading is the process of transmitting the binary or normal files to the server. Flask facilitates us to upload the files easily.
- It requires an HTML form whose enctype property is set to "multipart/form-data" to publish the file to the URL. The server-side flask script fetches the file from the request object using request.files[] Object. On successfully uploading the file, it is saved to the desired location on the server.
- Each uploaded file is first saved on a temporary location on the server, and then will actually be saved to its final location.
- The name of the target file can be hard-coded or available from the filename property of file request.files object. However, it is recommended that the secure_filename() function be used to obtain a secure version of it.
- The default upload folder path and maximum size of

uploaded files can be defined in the configuration settings for the Flask object.

- Define the path to the upload folder

```
app.config['UPLOAD_FOLDER']
```

- Specifies the maximum size (in bytes) of the files to be uploaded

```
app.config['MAX_CONTENT_PATH']
```

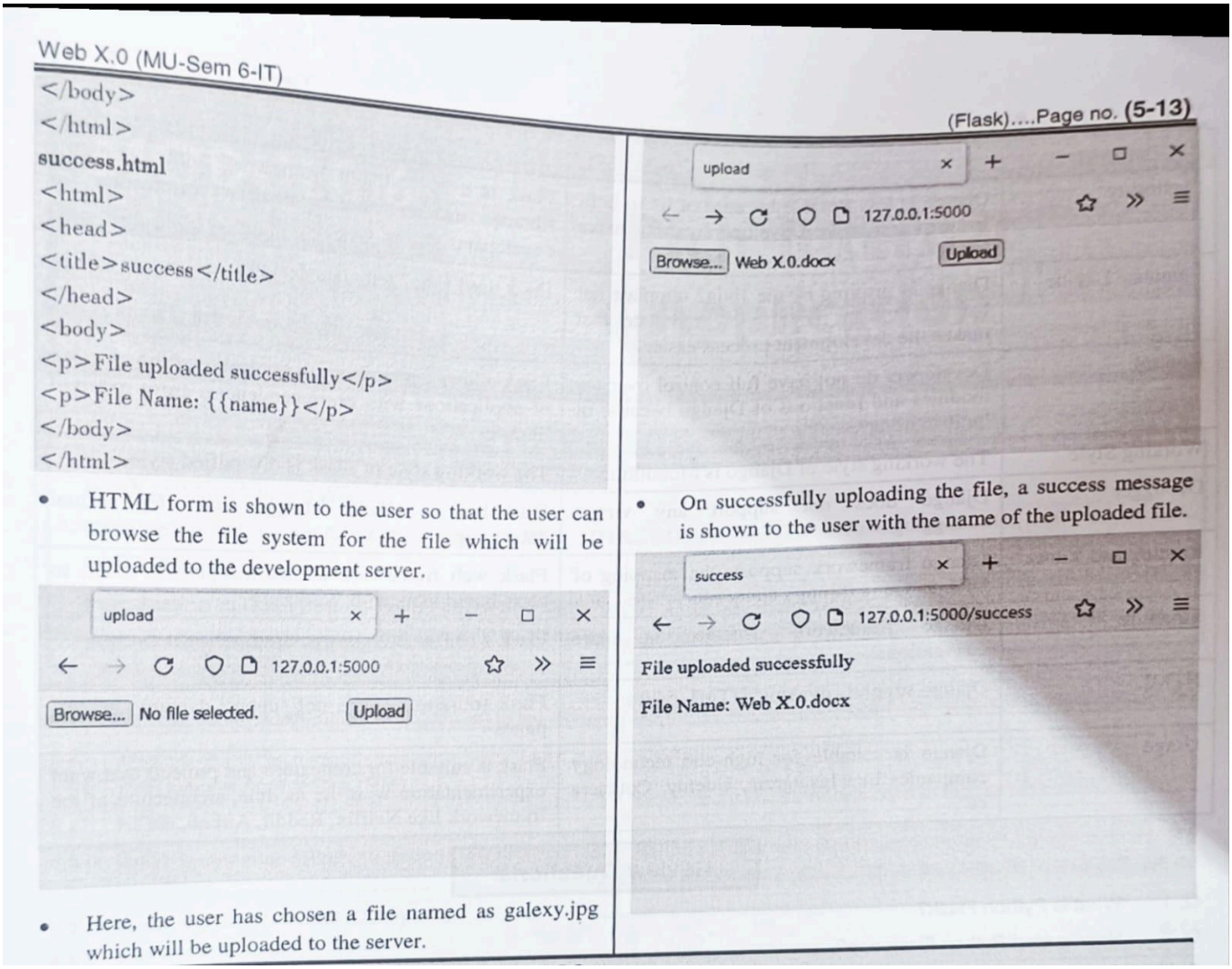
- **Example :** In this example, we will provide a file selector(file_upload_form.html) to the user where the user can select a file from the file system and submit it to the server.
- At the server side, the file is fetched using the request.files['file'] object and saved to the location on the server.
- Since we are using the development server on the same device, hence the file will be uploaded to the directory from where the flask script upload.py is executed.

upload.py

```
from flask import *
app = Flask(__name__)

@app.route('/')
def upload():
    return render_template("file_upload_form.html")

@app.route('/success', methods = ['POST'])
def success():
    if request.method == 'POST':
        f = request.files['file']
        f.save(f.filename)
        return render_template("success.html", name =
f.filename)
if __name__ == '__main__':
    app.run(debug = True)
```

file_upload_form.html

```
<html>
<head>
  <title>upload</title>
</head>
<body>
    <form action = "/success" method = "post"
enctype="multipart/form-data">
      <input type="file" name="file" />
      <input type = "submit" value="Upload">
    </form>
```



Q.7 What are the features of Python Flask?

Fast Debugging:

- Flask provides a built-in debugger that makes it easy to identify and fix errors in your application during development.
- When an error occurs, Flask's debugger provides detailed information about the issue, such as the stack trace, variables' values, and the line of code where the error occurred.
- This helps developers quickly identify and fix bugs, making the debugging process faster and more efficient.

RESTful Request Dispatching:

- Flask supports RESTful request dispatching, which means it allows you to design your application following REST (Representational State Transfer) principles.
- RESTful routing in Flask involves mapping HTTP methods (GET, POST, PUT, DELETE, etc.) to specific functions or views in your application.
- This makes it easier to create APIs that adhere to RESTful conventions, making your

application more organized and easier to maintain.

Application Programming Interface (API):

- Flask allows you to create APIs by defining routes that respond to HTTP requests with JSON or XML data.
- This makes it easy to build web services and APIs for your applications, enabling communication between different systems and allowing you to expose functionality to external clients or other parts of your application.

Flexible Configuration:

- Flask provides a flexible configuration system that allows you to customize various aspects of your application, such as debugging settings, database connections, secret keys, and more.
- You can use configuration files, environment variables, or Python code to configure your Flask application, making it easy to deploy and manage different configurations for development, testing, and production environments.

Integrated Unit Testing:

- Flask has built-in support for unit testing, which allows you to write tests to verify the functionality of your application's individual components (such as views, models, and forms) in isolation.
- Flask's testing framework provides tools for creating and running unit tests, making it easier to ensure that your application behaves as expected and catches bugs early in the development process.

Lightweight and minimalistic:

- Flask has very few dependencies and provides only the essential components for web development, such as routing, request handling, templating, and testing.
- This makes Flask easy to learn and use, and also gives you more flexibility and control over your application.

Jinja2 templating engine:

- Flask uses Jinja2 as its default templating engine, which lets you write dynamic HTML templates with Python-like syntax.
- Jinja2 supports features such as inheritance, macros, filters, and expressions, and also offers security features such as auto-escaping and sandboxing.

WSGI compliant:

- Flask is compliant with the Web Server Gateway Interface (WSGI) standard, which defines how web servers and web applications communicate.
- This means that Flask can run on any WSGI-compatible web server, such as Gunicorn, uWSGI, or Apache.

Extensible and modular:

- Flask supports a modular approach to web development, where you can organize your application into smaller and reusable components called blueprints.
- Flask also has a rich ecosystem of extensions that provide additional functionality, such as database integration, authentication, caching, email, and more. You can choose the extensions that suit your needs and customize them as you like

Routing

- Flask provides a simple yet powerful routing system that allows you to map URLs to view functions, making it easy to define the behavior of your application based on the incoming request URL.

Community

- Flask has a vibrant community of developers who contribute plugins, extensions, and tutorials to help other users get the most out of the framework. The Flask documentation is also comprehensive and well-maintained, making it easy to find answers to common questions and learn new features.

Q.8 What are the advantages of Python Flask?

Lightweight and Flexible:

- Flask is a lightweight web framework that allows developers to have more flexibility in building web applications.
- It doesn't come with all the bells and whistles of larger frameworks, which makes it easier to understand, customize, and extend according to specific project requirements.
- Flask follows a minimalistic approach, allowing developers to choose and integrate the tools and libraries they need, making it a great choice for small to medium-sized projects or for those who prefer a more modular approach.

2. Easy to Learn and Get Started:

- Flask has a simple and intuitive API that makes it easy for developers to get started quickly. It has a small learning curve, making it accessible to developers of all skill levels, including beginners.
- Flask provides clear documentation, tutorials, and a large community of users and developers, making it easy to find support and resources online.
- This makes Flask a great option for developers who want to quickly build web applications without spending too much time on complex setup or configuration.

3. Flexible Routing:

- Flask provides easy-to-use routing capabilities that allow developers to define URL routes for handling different HTTP requests.
- This makes it easy to define custom routes and handle various endpoints in the web application.

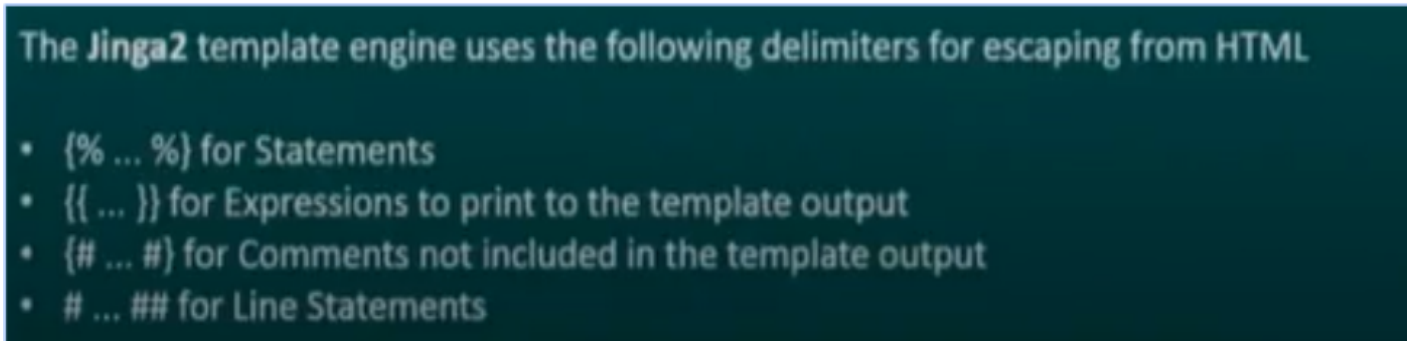
- Flask follows the WSGI (Web Server Gateway Interface) specification, which makes it compatible with a wide range of web servers, such as Gunicorn, uWSGI, and mod_wsgi. This flexibility in routing allows developers to design and structure their web applications in a way that suits their specific requirements.

4. Templating Engine:

- Flask comes with a built-in templating engine called Jinja2, which provides a powerful and flexible way to render dynamic HTML templates. Jinja2 allows developers to separate the presentation logic from the business logic, making it easier to maintain and update the application's views.
- Jinja2 provides features like template inheritance, filters, loops, and conditionals, making it a powerful tool for rendering dynamic content in web applications. Additionally, Flask allows developers to use other templating engines if preferred, giving them even more flexibility and choice.

(Ma'am Notes)

- Flask uses templates to expand the functionality of a web application while maintaining a simple and organized file structure.
- Templates are enabled using the Jinja2 template engine and allow data to be shared and processed before being turned into content and sent back to the client.
- Jinja template library to render templates
- Integrate some datasource to render it as a HTML page
- Feature
    - Sandbox execution mode.
    - Powerful automatic HTML escaping system for cross site scripting prevention.
    - Template inheritance which makes it possible to use a similar layout for all templates.
    - Easy to debug with a debugging system that integrates compile and runtime errors into standard Python traceback systems.
    - Optional Ahead of time template compilation.
- flask will try to find the HTML file In the template folder, in the same folder in which the script is present
- render_template() function renders HTML files for display in the web browser
- Handy when you need to create dynamic pages



5. Large Ecosystem:

Although Flask is a lightweight framework, it has a large ecosystem of extensions and libraries that can be easily integrated into Flask applications.

Q.9 How to set, access, and delete cookies in Python Flask?

- Cookies are stored on the client's computer as text files.
- Aim is to remember and track data that is relevant to customer usage for better visitor experience and website statistics.
- The Flask Request object contains the properties of the cookie.
- It is a dictionary object for all cookie variables and their corresponding values, and the client is transferred.In addition to this, cookies also store the expiration time, path, and domain name of its website.
- Cookies are set on the response object.
    - Server sends the Cookie to the user along with the response. It is done using the make_response() function.
    - Once the response is set, we use the set_cookie() function to attach the cookie to it.
- The cookie takes the attributes:
  response.set_cookie('<Title>','<Value>','<Expiry Time>')
- The cookie is sent back along with the Request to the server.
    - to get the Cookie back from the user, use request.cookies.get() function

gfg

Setting Cookies in Flask:

set_cookie( ) method: Using this method we can generate cookies in any application code. The syntax for this cookies setting method:

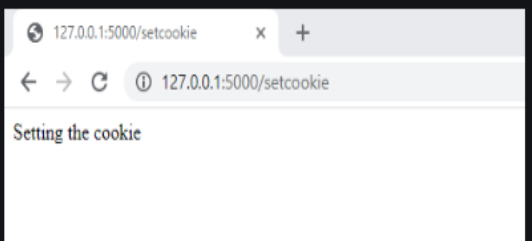
```
Response.set_cookie(key, value = '', max_age = None, expires = None, path = '/', domain = None,
                    secure = None, httponly = False)
```

Parameters:

- key – Name of the cookie to be set.
- value – Value of the cookie to be set.
- max_age – should be a few seconds, None (default) if the cookie should last as long as the client's browser session.
- expires – should be a datetime object or UNIX timestamp.
- domain – To set a cross-domain cookie.
- path – limits the cookie to given path, (default) it will span the whole domain.

```python
from flask import Flask, request, make_response

app = Flask(__name__)

# Using set_cookie( ) method to set the key-value pairs below.
@app.route('/setcookie')
def setcookie():

      # Initializing response object
    resp = make_response('Setting the cookie')
    resp.set_cookie('GFG','ComputerScience Portal')
    return resp

app.run()
```

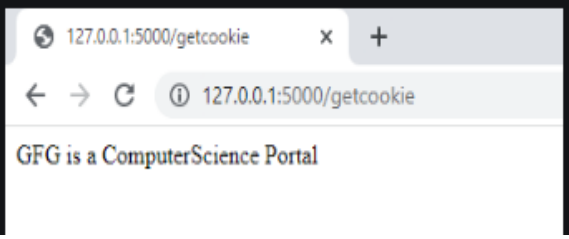
Output: Go to the above-mentioned url in the terminal -For Example – http://127.0.0.1:5000/route-name. Here the route-name is setcookie.



Getting Cookies in Flask:
cookies.get( )
This get( ) method retrieves the cookie value stored from the user's web browser through the request object.

```python
from flask import Flask, request, make_response
app = Flask(__name__)

# getting cookie from the previous set_cookie code
@app.route('/getcookie')
def getcookie():
    GFG = request.cookies.get('GFG')
    return 'GFG is a '+ GFG

app.run()
```

Output:



delete cookies:

To delete a cookie in Python Flask, you can use the delete_cookie method provided by Flask's response object.

code:

```
from flask import Flask, make_response

app = Flask(__name__)

@app.route('/')
def delete_cookie():
    response = make_response('Cookie deleted!')
    response.delete_cookie('cookie_name')
    return response

if __name__ == '__main__':
    app.run()
```

This code creates a route that, when accessed, deletes the specified cookie and returns a response saying "Cookie deleted!"

Q.10 Differentiate between Flask and Django frameworks.

Django	Flask
Django could be a Python-based free, open-source system that takes after the MVT(model view Template) approach of structural design	Flask could be a Python-based smaller-scale system without any set of specific instruments or outside libraries. It too doesn't have a database layer or arrangements for shape approval and makes utilize of expansions.

Urls.py is utilized to set the association properties and demands are handled by the primary coordinating see of the regex list	URI is most regularly than not set by the see decorator and centralized setup is additionally conceivable. Sometimes the recent designs are coordinated with the URIs, and the last mentioned is sorted in a default arrange
Doesn't exclude setting flexibility	It is accepted that all the conceptual stages to organize a Flask code rise and smaller-scale open-source to the applications number show in Flask as of now
Extend Layout is a Conventional extended structure	Extend Layout is an Arbitrary structure
Django gives an all-inclusive encounter: you get an admin board, database interfacing, an ORM, and a registry structure for your apps and ventures out of the box.	Flask gives straightforwardness, adaptability, and fine-grained control. It is unopinionated
Django provides built-in authentication and authorization systems, making it easier to implement user management and access control.	Flask leaves authentication and authorization to the developer. You can use third-party libraries like Flask-Login and Flask-Principal for these functionalities.

It is suitable for multi-page applications.	It is suitable for single-page applications only.
Its framework structure is more conventional.	Random web framework structure.
Django has a large and active community, which means access to extensive documentation, tutorials, and third-party packages. It is widely used in various industries.	Flask also has a thriving community, but it may have fewer resources compared to Django. Flask's simplicity attracts developers who prefer to keep their stack minimal.
It doesn't support any virtual debugging.	It has a built-in debugger that provides virtual debugging.
Its working style is Monolithic.	Its working style is diversified style.

GPT

Feature	Django	Flask
Architecture	Follows the Model-View-Template (MVT)	Follows a lightweight, microframework design
Philosophy	"Batteries included" - emphasizes convention over configuration	"Lightweight" - emphasizes flexibility and simplicity

Database Support	**Built-in ORM with support for multiple databases**	**Supports various databases but relies more on third-party extensions**
Admin Interface	**Provides a powerful built-in admin interface**	**Minimalistic admin interface that can be extended with Flask extensions**
Template Engine	**Built-in template engine**	**Uses Jinja2 as the default template engine**
URL Routing	**Uses a built-in URL dispatcher**	**Uses decorators for routing**
Form Handling	**Includes form handling and validation**	**Requires third-party extensions for form handling**
Authentication	**Built-in authentication system**	**Requires third-party extensions for authentication**
Security	**Offers built-in security features**	**Relies on third-party extensions for security features**
REST API Support	**Provides support for building APIs through third-party packages**	**Offers support for building APIs out-of-the-box**
Learning Curve	**Steeper learning curve due to its complexity**	**Easier learning curve due to its simplicity**
Community & Ecosystem	**Large community with extensive documentation and plugins**	**Active community with a variety of extensions and resources**
Scalability	**Well-suited for large, complex applications**	**More flexible and can be used for smaller projects or scaled up as needed**
Suitability	**Ideal for large-scale applications with complex requirements**	**Ideal for small to medium-sized projects with simpler requirements**
Deployment	**Requires more setup for deployment**	**Easier deployment due to its lightweight nature**

Q.11 Why is Flask called a microframework?

- Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.
- It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.
- However, Flask supports extensions that can add application features as if they were implemented in Flask itself.
- Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies, and several common framework-related tools

OR

Flask is aptly referred to as a micro web framework because of its minimalist design and lightweight nature. Here's why it earns this classification:

1. **Minimal Dependencies:** Flask does not require specific tools or libraries to function. Unlike larger frameworks, it doesn't impose a rigid structure or force you to use particular components. You can start building a Flask application with just the essentials.
2. **No Built-in Database Abstraction Layer:** Unlike some other frameworks, Flask doesn't include a built-in database abstraction layer. Instead, it allows you to choose your preferred database library or ORM (Object-Relational Mapper) based on your project's needs.
3. **No Form Validation or Pre-existing Components:** Flask doesn't come bundled with features like form validation, authentication, or other common functionalities. Instead, it encourages developers to use third-party libraries for these purposes. This approach keeps the core of Flask simple and flexible.
4. **Extensible via Extensions:** While Flask itself remains minimal, it provides a robust extension system. Developers can add features to their Flask applications by incorporating extensions. These extensions seamlessly integrate with Flask, making it feel as if they were part of the framework itself.

Module 6: RIA

Q.1 Define RIA and explain the characteristics of RIA

Rich Internet applications (RIA) are Web-based applications that have some characteristics of graphical desktop applications.

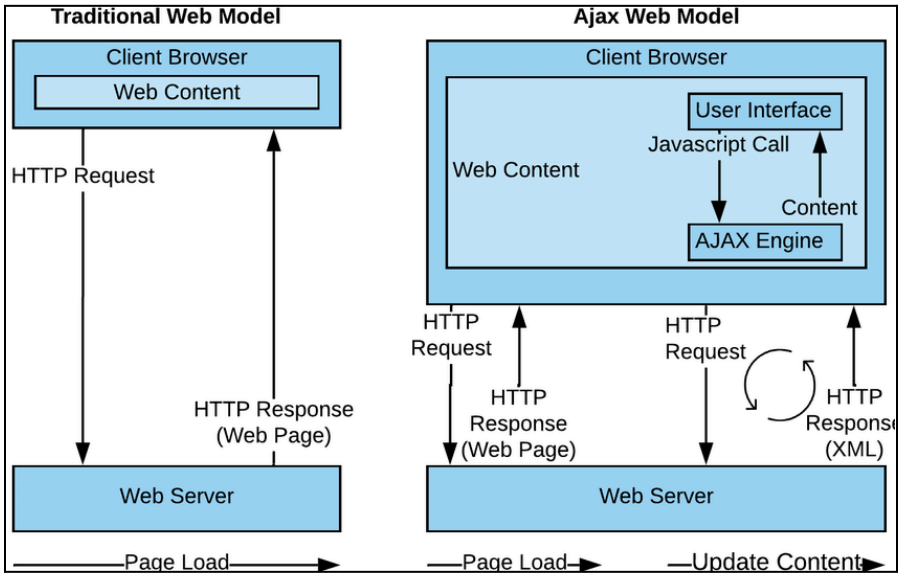
- Built with powerful development tools, RIAs can run faster and be more engaging.
- They can offer users a better visual experience and more interactivity than traditional browser applications that use only HTML and HTTP.
- RIAs typically leverage advanced web technologies such as AJAX (Asynchronous JavaScript and XML), HTML5, CSS3, and JavaScript to provide features like interactivity, responsiveness, multimedia content, and offline capabilities.

Characteristics of Rich Internet Applications include:

- **Interactivity**: RIAs allow users to interact with the application in real-time without needing to reload the entire webpage. This enables features like drag-and-drop, dynamic content updates, and form validation without page refreshes.
- **Rich Media**: RIAs can incorporate rich media elements such as audio, video, animations, and graphics to enhance the user experience and engagement.
- **Asynchronous Data Loading:** RIAs use techniques like AJAX to fetch data from the server asynchronously, enabling faster response times and a smoother user experience by updating only the necessary parts of the page.

- **Cross-Platform Compatibility:** RIAs are designed to run on multiple platforms and devices, including desktop computers, laptops, tablets, and smartphones, without requiring platform-specific development.
- **Responsive Design**: RIAs can adapt to different screen sizes and orientations, providing a consistent user experience across devices through responsive design principles.
- **Offline Capabilities**: Some RIAs can continue to function even when the user is offline by leveraging technologies like local storage and caching. This allows users to access certain features and content without an active internet connection.
- **Client-Side Processing**: RIAs offload some processing tasks from the server to the client's browser, improving performance and reducing server load. This is often achieved using client-side scripting languages like JavaScript.
- **Integration with Web Services**: RIAs seamlessly integrate with web services and APIs to access and manipulate data from external sources, enabling features like social media integration, real-time updates, and third-party content embedding.
- **Security**: RIAs incorporate security features to protect user data and prevent unauthorized access, including encryption, secure communication protocols, input validation, and authentication mechanisms.

Q.2 Explain Working AJAX



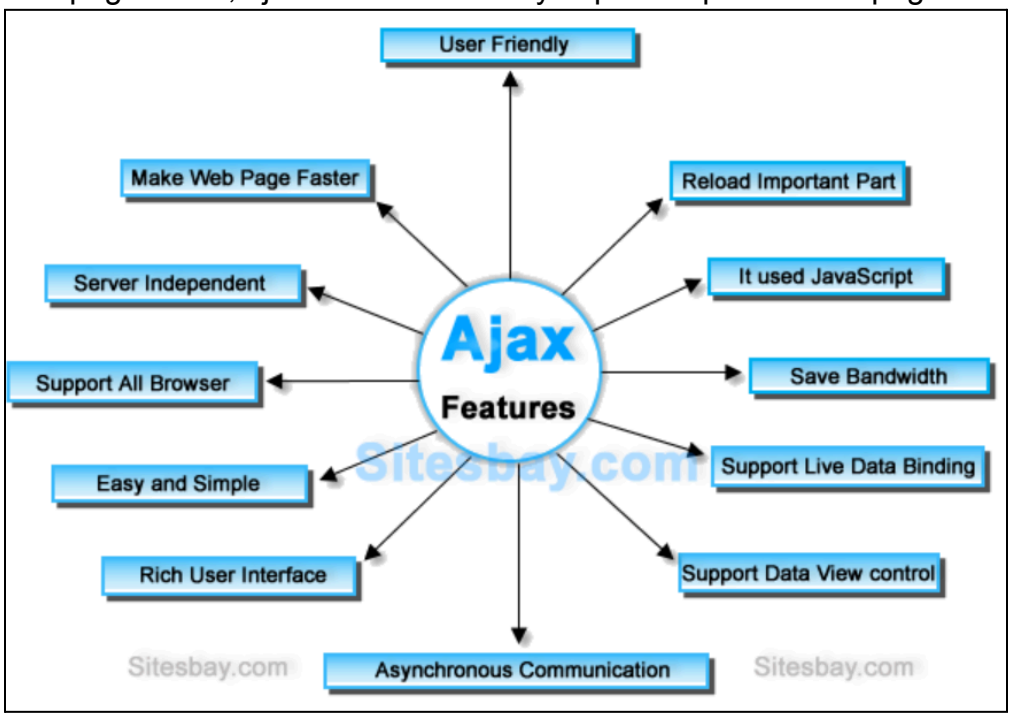
Here's how AJAX typically works:

- Client-Side JavaScript: AJAX is implemented using JavaScript, which is executed on the client-side (in the user's web browser). JavaScript is responsible for making asynchronous requests to the server and handling the response.
- XMLHttpRequest (XHR) Object: In traditional AJAX, the XMLHttpRequest object is used to interact with the server asynchronously. However, modern web development often uses the fetch() API for making AJAX requests, which provides a more modern and flexible approach to handling asynchronous requests.

- Sending Requests: To fetch data from the server or send data to the server, JavaScript code constructs an AJAX request and sends it to the server. This request can be an HTTP GET, POST, PUT, DELETE, or other HTTP methods, depending on the type of operation required.
- Server-Side Processing: On the server-side, the server processes the AJAX request, performs the necessary operations (such as retrieving or updating data), and generates a response.
- Handling Responses: Once the server processes the request, it sends a response back to the client. The response can be in various formats, such as HTML, JSON, XML, or plain text, depending on the server's configuration and the nature of the data being exchanged.
- Updating the DOM: When the client receives the response from the server, JavaScript code processes the response and updates the DOM (Document Object Model) dynamically to reflect the changes. This could involve updating specific elements on the web page, adding new content, or performing other actions based on the data received from the server.

Q.3 Discuss features of AJAX.

Ajax is an acronym for Asynchronous JavaScript and XML. Asynchronous means that the user need not wait until the server replies. It is a combination of different technologies like JavaScript, DOM, XML, HTML, CSS etc, used for building rich and responsive user interfaces. The main Feature of Ajax is it make web-page faster, ajax allow reload only important part of web-page not complete website or web-page.



1. **User-Friendly:** Ajax enhances the user experience by providing seamless and interactive web applications. Users can perform actions without waiting for the entire page to reload, leading to a more responsive interface.
2. **Faster Web Page:** Ajax allows for asynchronous communication with the server, enabling parts of the web

page to be updated dynamically without requiring a full page reload. This leads to faster loading times and a smoother browsing experience for users.

3. **Independent of Server Technology:** Ajax is independent of server-side technologies. It can be used with any server-side language (e.g., PHP, Node.js, Python) and any database technology (e.g., MySQL, MongoDB), making it versatile and widely applicable.
4. **Increased Performance**: By reducing the need for full page reloads, Ajax significantly improves the performance of web applications. Only the necessary data is exchanged between the client and server, resulting in faster response times.
5. **Support for Live Data Binding:** Ajax supports live data binding, allowing changes made to data on the server to be reflected in the web page in real-time without requiring manual refreshes.
6. **Support for Data View Control**: Ajax provides support for data view controls, allowing developers to present data in various formats (e.g., tables, grids) dynamically on the web page.
7. **Support for Client-Side Template Rendering:** Ajax enables client-side template rendering, allowing developers to define templates for displaying data on the client side. This enhances the flexibility and customization of web applications.
8. **Rich and Responsive User Interfaces:** Ajax facilitates the creation of rich and responsive user interfaces by enabling dynamic updates and interactions without page refreshes. This results in a more engaging and interactive user experience.
9. **Reduced Consumption of Server Resources**: Since Ajax requests only fetch the necessary data from the server, it reduces the load on the server and conserves server resources. This scalability makes Ajax suitable for handling high traffic volumes.
10. **No Need for Full Page Reloads**: With Ajax, there is no need to push a submit button and reload the entire website. Only the specific parts of the page that need updating are reloaded, leading to a smoother and more efficient browsing experience.
11. **Cross-Browser Compatibility:** Apart from obtaining the XMLHTTP object, Ajax processing is the same for all browser types, as JavaScript is used universally. This ensures consistent behavior across different browsers.
12. **Faster Development**: Ajax simplifies the development process by providing a more efficient way to build interactive web applications. Developers can focus on creating dynamic features without worrying about page reloads and server round-trips.
13. **Bandwidth Efficiency:** Ajax reduces the need to reload entire web pages, resulting in lower bandwidth consumption. Only the necessary data is transmitted between the client and server, optimizing bandwidth usage.

Q.4 Discuss DOM

Q.5 Discuss XML HTTP Request, its methods & properties.

- XMLHttpRequest object is an API that is used for fetching data from the server.
- XMLHttpRequest is used in Ajax programming.
- It retrieves any type of data such as JSON, XML, text, etc.
- It requests data in the background and updates the page without reloading the page on the client side.

- An object of XMLHttpRequest is used for asynchronous communication between the client and server.
- **Properties of XMLHttpRequest:**
    a. **onload**: When the request is received (loaded), it defines a function to be called.
    b. **onreadystatechange**: A function will be called whenever the readyState property changes.
    c. **readyState**: It defines the current state of the request or holds the current status of the XMLHttpRequest. There are five states of a request:
        i. readyState= 0: It represents the Request not initialized.
        ii. readyState= 1: Establishment of server connection.
        iii. readyState= 2: Request has been received
        iv. readyState= 3: During the time of processing the request
        v. readyState= 4: Response is ready after finishing the request
    d. **responseText**: It will return the data received by the request in the form of a string.
    e. **responseXML**: It will return the data received by the request in the form of XML data.
    f. **status**: It will return the status number of the request. (i.e. 200 and 404 for OK and NOT FOUND respectively).
    g. **statusText**: It will return the status text in the form of a string. (i.e. OK and NOT FOUND for 200 and 404 respectively).
- **Methods of XMLHttpRequest:**

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(*method,url,async,user,psw*)	Specifies the request *method*: the request type GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous *user*: optional user name *psw*: optional password
send()	Sends the request to the server Used for GET requests
send(*string*)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

Q.6 Explain the features of Django along with the architecture.

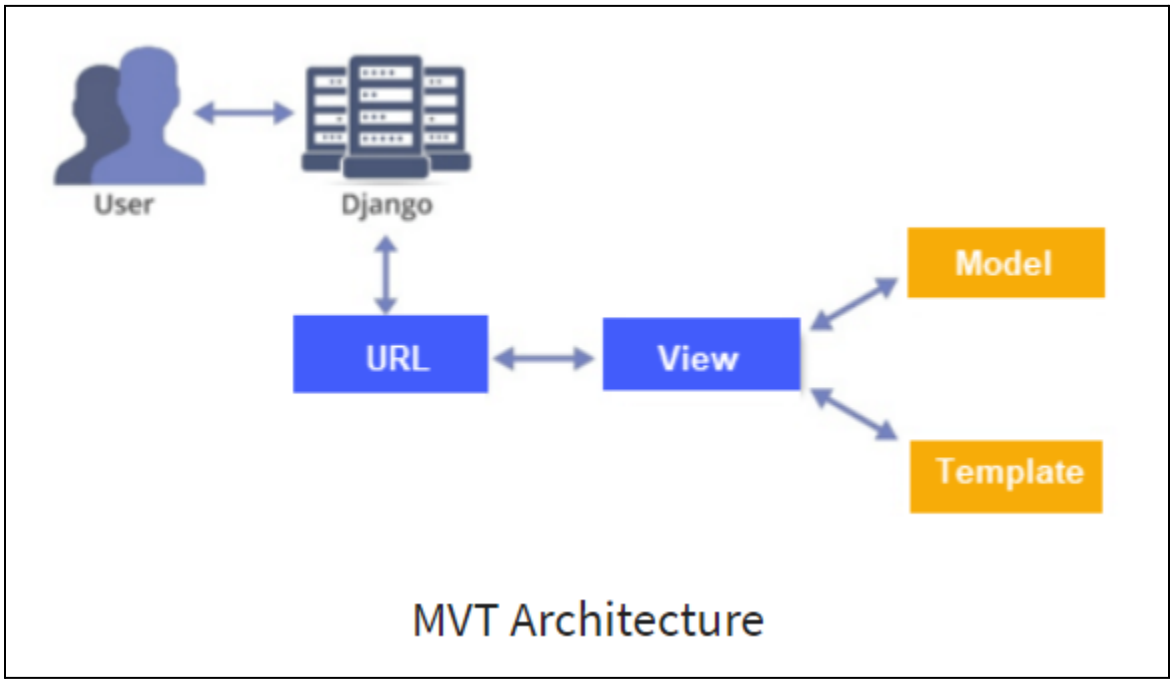
Django is a high-level Python web framework that enables rapid development of secure and maintainable websites and web applications. It follows the Model-View-Template (MVT) architectural pattern, which is a variation of the more common Model-View-Controller (MVC) pattern.

Architecture of Django (MVT pattern):

Model: The Model layer defines the data structure of your application. Django models are Python classes that define the fields and behaviors of the data you're storing. Models map to database tables, and Django's ORM handles the interactions between your Python code and the database.

View: The View layer is responsible for processing requests and returning responses. Views are Python functions or classes that receive web requests, perform any necessary logic (such as querying the database), and return an HTTP response (often generated using a template).

Template: The Template layer is responsible for generating HTML output to be sent back to the client. Templates are HTML files that can include Python-like syntax (using Django's template language) for inserting dynamic content generated by views.



Features of Django:

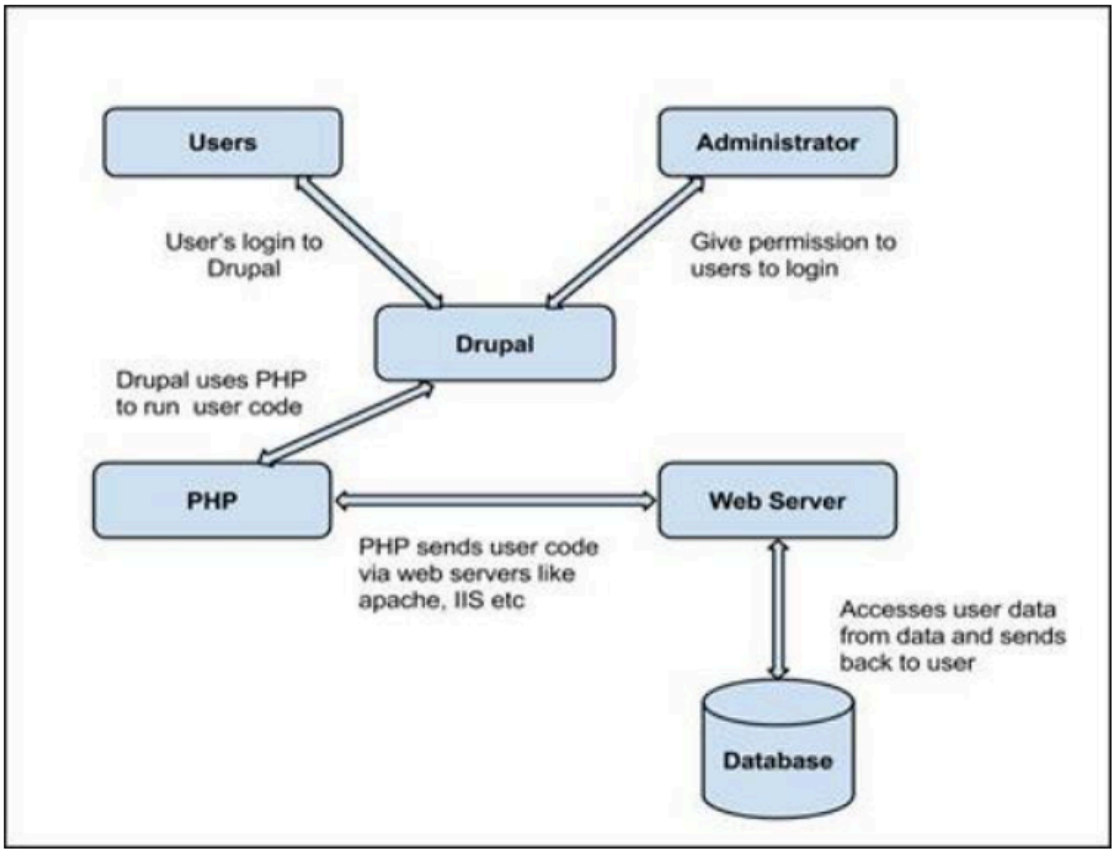
- Object-Relational Mapping (ORM): Django provides a powerful ORM that maps database tables to Python objects, making database operations more Pythonic and abstracting away the complexities of SQL queries.
- Admin Interface: Django comes with a built-in admin interface that allows developers to manage site content through a user-friendly interface, without having to write additional code.
- URL Routing: Django uses a URL dispatcher to direct incoming web requests to the appropriate view based on the requested URL pattern.
- Template Engine: Django includes a template engine that allows developers to create dynamic HTML templates using Python-like syntax, making it easy to build interactive web pages.
- Forms Handling: Django provides a forms library that simplifies the process of creating and validating HTML forms, including handling form submission and data cleaning.
- Authentication and Authorization: Django includes robust authentication and authorization features, including user authentication, permissions, and groups, making it easy to secure your application.
- Security Features: Django provides built-in protection against common security threats such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).
- Internationalization and Localization: Django supports internationalization (i18n) and localization (l10n) features, making it easy to create multilingual websites.

- Caching: Django includes a caching framework that allows developers to cache dynamic content to improve performance and reduce database load.
- Testing Framework: Django provides a testing framework that makes it easy to write and run tests for your application, ensuring its reliability and robustness.

Q.7 Explain the features of Drupal along with the architecture.

Architecture:

Drupal is a platform for web content management which is a powerful tool for building simple and complex sites.Following is the architectural style of Drupal for implementing user interfaces. The following diagram shows the architecture of Drupal –



Drupal Taxonomies

The architecture of Drupal contains the following layers;

Users – These are the users on the Drupal community. The user sends a request to a server using Drupal CMS and web browsers, search engines, etc. acts like clients.

Administrator – Administrator can provide access permission to authorized users and will be able to block unauthorized access. Administrative account will be having all privileges for managing content and administering the site.

Drupal - Drupal is a free and open source Content Management System (CMS) that allows organizing, managing and publishing your content and is built on PHP based environments. Drupal CMS is very flexible and powerful and can be used for building large, complex sites. It is very easy to interact with other sites and technologies using Drupal CMS. Further, you will be able to handle complex forms and workflows.

PHP - Drupal uses PHP in order to work with an application which is created by a user. It takes the help of web server to fetch data from the database. PHP memory requirements depend on the modules which are used in your site. Drupal 6 requires at least 16MB, Drupal 7 requires 32MB and Drupal 8 requires 64MB.

Web Server - Web server is a server where the user interacts and processes requests via HTTP (Hyper Text Transfer Protocol) and serves files that form web pages to web users. The communication between the user and the server takes place using HTTP. You can use different types of web servers such as Apache, IIS, Nginx, Lighttpd, etc.

Database - Database stores the user information, content and other required data of the site. It is used to store the administrative information to manage the Drupal site. Drupal uses the database to extract the data and enables to store, modify and update the database.

Features:

- Drupal makes it easy to create and manage your site.
- Drupal translates anything in the system with built-in user interfaces.
- Drupal connects your website to other sites and services using feeds, search engine connection capabilities, etc.
- Drupal is an open source software hence requires no licensing costs.
- Drupal designs highly flexible and creative website with effective display quality thus increasing the visitors to the site.
- Drupal can publish your content on social media such as Twitter, Facebook and other social mediums.
- Drupal provides more number of customizable themes, including several base themes which are used to design your own themes for developing web applications.
- Drupal manages content on informational sites, social media sites, member sites, intranets and web applications.

Q.8 Explain the features of Joomla along with the architecture.

Q.9 Explain the Asynchronous and synchronous working model.

Synchronous (Classic Web-Application Model)

The synchronous model can be classified into three stages :

(1) On the client side, if the user does any activity on the browser, the browser sends a HTTP request to the server and waits for a response.

(2) The server does some processing on the request of the user and sends a response to the user.

(3) When the response is received at the user end, the browser reloads the webpage to show the information.

Fig. 6.1.1 illustrate the classical model or a synchronous model.

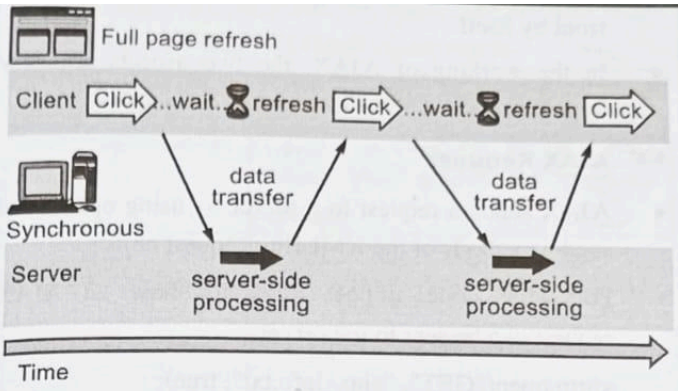


Fig. 6.1.1 : Synchronous Model

- Fig. 6.1.1 depicts a synchronous model, where the request blocks the client until the particular operation completes such as wait or refresh.
- Fig. 6.1.2 shows the process and format of data transfer involved in a synchronous model.

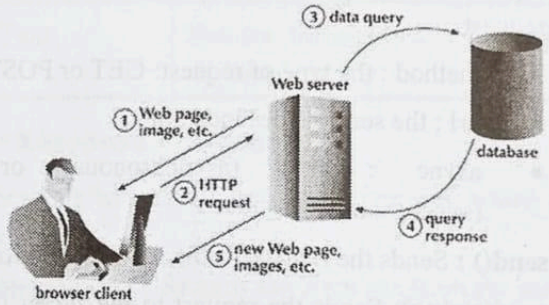


Fig. 6.1.2 : Process involved in the synchronous model

- In Fig. 6.1.2, the browser directly sends a request to the server in the form of an HTTP request. After processing the request, the server sends the response to the browser in the form of HTML and CSS data.

Asynchronous (AJAX Web-Application Model)

- AJAX asynchronous model eliminates the waiting process of the client after each event, by introducing an AJAX engine between the user and the server. This AJAX communicates with the server on the user's behalf.
- Fig. 6.1.3 shows the AJAX asynchronous model.

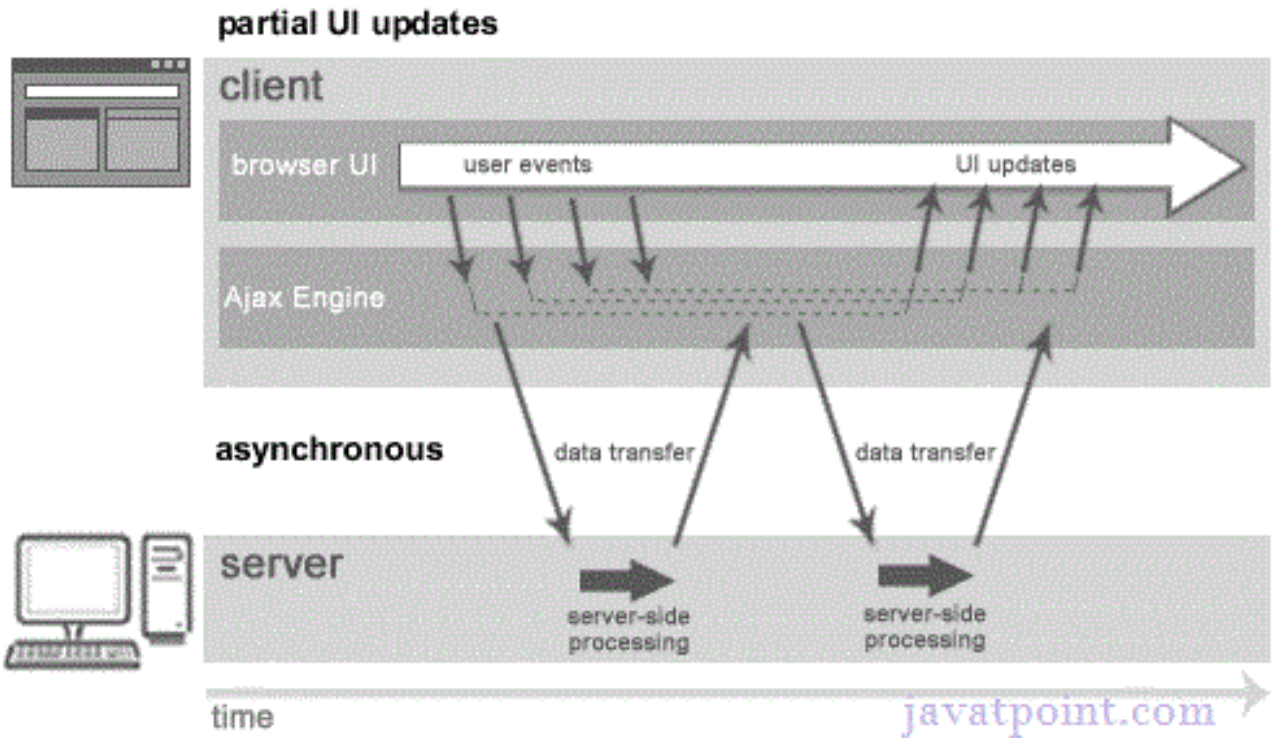


Fig. 6.1.3 AJAX Asynchronous Model

Fig. 6.1.3 : AJAX Asynchronous model

- Fig. 6.1.3 illustrate the AJAX asynchronous model, where the request doesn't block the client. This means that the browser responds to the client's request instantaneously. Fig. 6.1.4 depicts the process involved in AJAX asynchronous model.

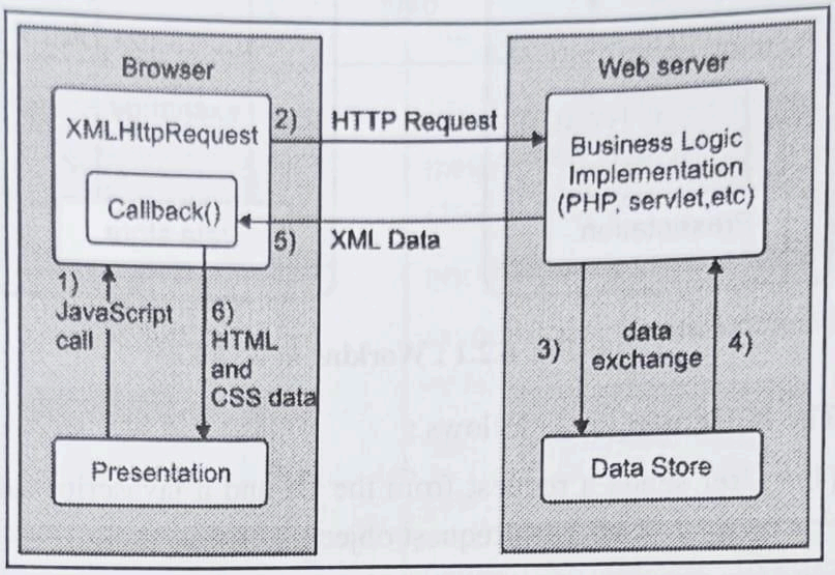


Fig. 6.1.4 : Process involved in AJAX Asynchronous model

- An AJAX asynchronous model generates a user action by invoking JavaScript call.
- This JavaScript call is handled by the AJAX engine, which converts this call to XMLHttpRequest. AJAX engine sends request to the server.
- The server does some processing according to the request and sends a response to the AJAX engine in the form of XML data. AJAX engine receives XML data and converts it to HTML and CSS and shows it on the webpages.

Q.10 Difference between CMS & Framework

Attribute	CMS	Framework
Definition	A Content Management System (CMS) is a software application that allows users to create, manage, and modify digital content.	A Framework is a collection of pre-written code and tools that provide a foundation for building software applications.
Primary Purpose	To manage and publish content on websites or web applications.	To provide a structure and set of tools for developing software applications.
Content Editing	Allows non-technical users to easily create and edit content using a user-friendly interface.	Does not provide built-in content editing capabilities. Developers need to implement content management functionality separately.
Customization	Often provides a wide range of themes, plugins, and extensions to customize the appearance and functionality of the CMS.	Allows developers to customize every aspect of the application according to their specific needs.
Learning Curve	Generally easier to learn and use for non-technical users due to its user-friendly interface.	Requires a higher level of technical expertise and programming knowledge to effectively use and develop applications using the framework.
Flexibility	Provides a high level of flexibility in managing and organizing content.	Offers flexibility in terms of application architecture and design, allowing developers to choose the components and libraries they prefer.
Scalability	Can handle large amounts of content and traffic, making it suitable for enterprise-level websites.	Can be highly scalable depending on the chosen framework and architecture.
Security	Often includes built-in security features and regular updates to address vulnerabilities.	Security measures need to be implemented by developers based on the specific requirements of the application.