Question 1

Confidentiality, integrity, and availability, often known as the CIA triad, are the building blocks of information security. Any attack on an information system will compromise one, two, or all three of these components. Based on which of these components is being compromised the most, efficient security controls can be designed accordingly.

Confidentiality

In simple terms, confidentiality means something that is secret and is not supposed to be disclosed to unintended people or entities. Confidentiality ensures that sensitive information is accessed only by an authorized person and kept away from those not authorized to possess them. Everyone has information that they wish to keep secret. Thus Protecting such information is an important part of information security.

Integrity

In the context of the information security (InfoSec) world, integrity means that when a sender sends data, the receiver must receive exactly the same data as sent by the sender. Data must not be changed in transit. For example, if someone sends a message "Hello!", then the receiver must receive "Hello!" That is, it must BE exactly the same data as sent by the sender. Any addition or subtraction of data during transit would mean the integrity has been compromised.

Availability

Availability implies that information is available to the authorized parties whenever required. Unavailability of data and systems can have serious consequences. It is essential to have plans and procedures in place to prevent or mitigate data loss as a result of a disaster. A disaster recovery plan must include unpredictable events such as natural disasters and fire. A routine backup job is advised in order to prevent or minimize total data loss from such occurrences. Also, extra security equipment or software such as firewalls and proxy servers can guard against downtime and unreachable data due to malicious actions such as denial-of-service (DoS) attacks and network intrusions.

So according to the definitions stated above:

a) **Distributed denial of service attack - Availability**
This is a compromise of **availability** because as a result of the DDOS attacks a huge number of fake requests are generated to the web servers due to which there is a lack of availability for legitimate customers.

b) **Ransomware - Availability**
Demanding money to restore access to systems and data targets the "A" that stands for **Availability** in CIA, the classic security triad of Confidentiality, Integrity, and Availability.

c) **Phishing - Confidentiality**
Phishing is a type of social engineering attack often used to steal user data, including login credentials and credit card numbers which compromises the confidentiality of the victim. It occurs when an attacker, masquerading as a trusted entity, dupes a victim into opening an email, instant message, or text message.

d) **Unauthorized Bitcoin mining on laptop** - Availability, Integrity
Cryptojacking is malicious cryptomining that happens when cybercriminals hack into both business and personal computers, laptops, and mobile devices to install the software. This software uses the computer's power and resources to mine cryptocurrencies or steals cryptocurrency wallets owned by unsuspecting victims. The

code is easy to deploy, runs in the background, and is difficult to detect. This hence compromises the availability of resources of the victims and the integrity of the crypto mining is also compromised.

e) **Ad Fraud - Integrity**
There are a variety of ways that cybercriminals can carry out ad fraud. Some of the methods include. Hidden ads: When an ad is shown in such a way that the user doesn't actually see it. This kind of fraud targets ad networks that pay based on impressions (views), not clicks. Click hijacking: This is when an attacker redirects a click on one ad to be a click for a different ad, effectively "stealing" the click. For this fraud attack to work, the attacker has to compromise the user's computer, the ad publisher's website, or a proxy server. Fake app installation: Ads are often shown within applications, especially mobile apps. For this fraud method, teams of people (often in click farms*) install apps thousands of times and interact with them in bulk. Botnet ad fraud: Scammers can use botnets to generate thousands of fake clicks on an ad or fake visits to a website displaying the ads. See below for more on how this works. This kind of attack compromises the integrity of the ad provider which is because the users are lead to different websites on clicking.

f) Bots grabbing vaccine appointments - Availability
This kind of attack is very similar to DDOS attacks n which the legitimate users who want the vaccines are denied it because of the fake vaccine appointments created by bots. So the availability is compromised

Question 2

a) **Email**
The history of modern Internet email services reaches back to the early ARPANET, with standards for encoding email messages published as early as 1973 (RFC 561). An email message sent in the early 1970s is similar to a basic email sent today. The Latest used is RFC 5321 — Simple Mail Transfer Protocol. This is a protocol used to send emails between computers published in October 2008.

b) **Web**
RFC 1, titled "Host Software", was written by Steve Crocker of the University of California, Los Angeles (UCLA), and published on April 7, 1969. HTTP/1.1 was first documented in RFC 2068 in 1997, and as of 2021, it (plus older versions) is less popular (used by less than 45% of websites; it's always a backup protocol) for web serving than its successors. That specification was obsoleted by RFC 2616 in 1999, which was likewise replaced by the RFC 7230 family of RFCs in 2014.

c) **File transfer**
The early version of FTP, with revisions published as RFC 172 in June 1971 and RFC 265 in November 1971. The first major revision was RFC 354, July 1972, which for the first time contained a description of the overall communication model used by modern TCP and details on many of the current features of the protocol. The latest used is RFC 959 - File Transfer Protocol - IETF Tools which was released in October 1985.

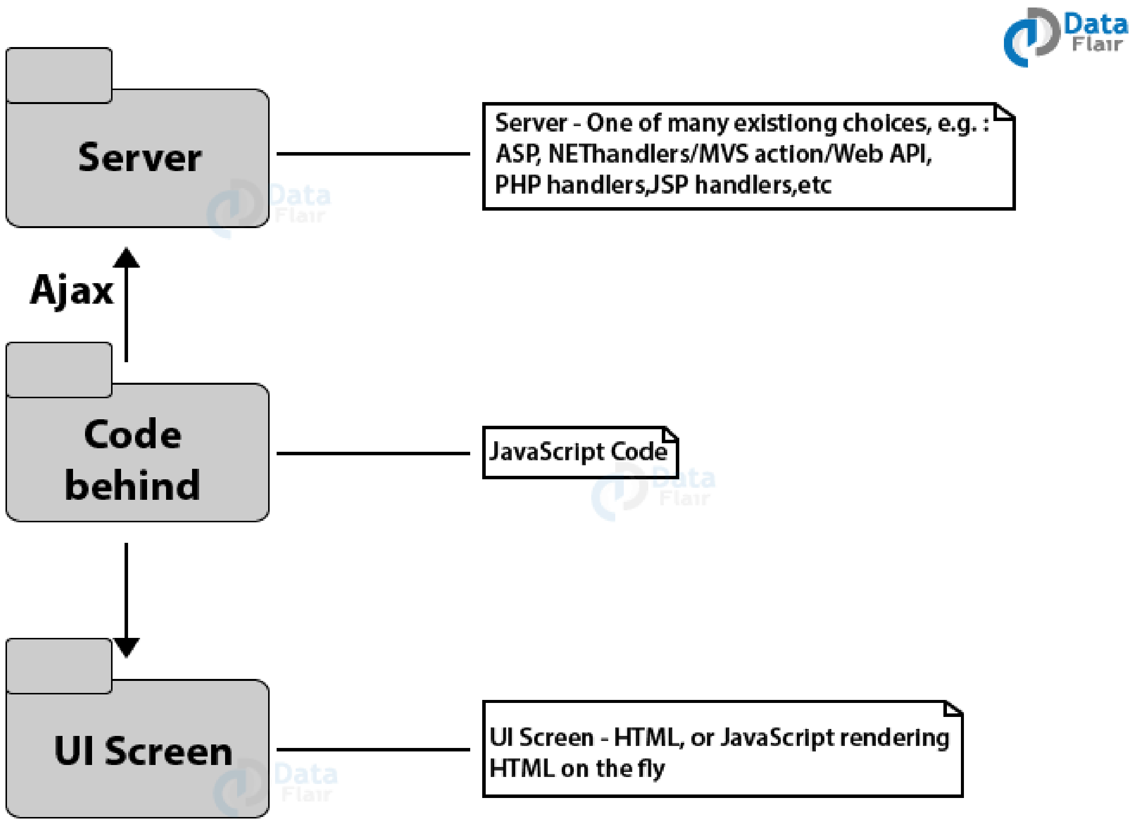
d) **Voice over IP**
The first protocol to be widely used for VoIP was H323. However, this is not a particularly rigorous definition and as a result, other variants have been developed. Megaco This is also

known as IETF RFC 2885 and ITU Recommendation H. 248. Modern VoIP used are RFC 6405 - Voice over IP (VoIP) SIP Peering Use Cases and RFC 7340 - Secure Telephone Identity Problem Statement which was released in STIR Problem Statement in September 2014.

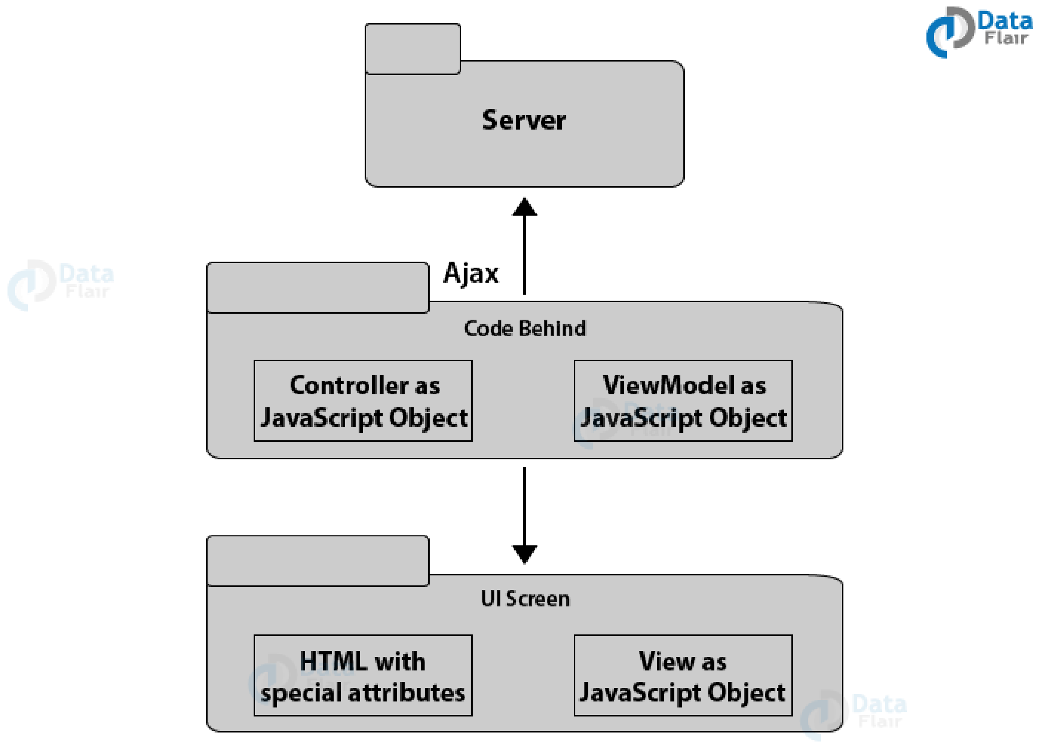
e) **Remote access to computing resources**
The simplest protocol that was designed to access remote computers over a network is probably telnet RFC 854. The current version is the RFC 4251 - The Secure Shell (SSH) Protocol Architecture. This was proposed in RFC 4251 SSH Protocol Architecture in January 2006.

Question-3

a)


Data Flair
Server
Server - One of many existiong choices, e.g. : ASP, NEThandlers/MVS action/Web API, PHP handlers,JSP handlers,etc
Ajax
Code behind
JavaScript Code
UI Screen
UI Screen - HTML, or JavaScript rendering HTML on the fly

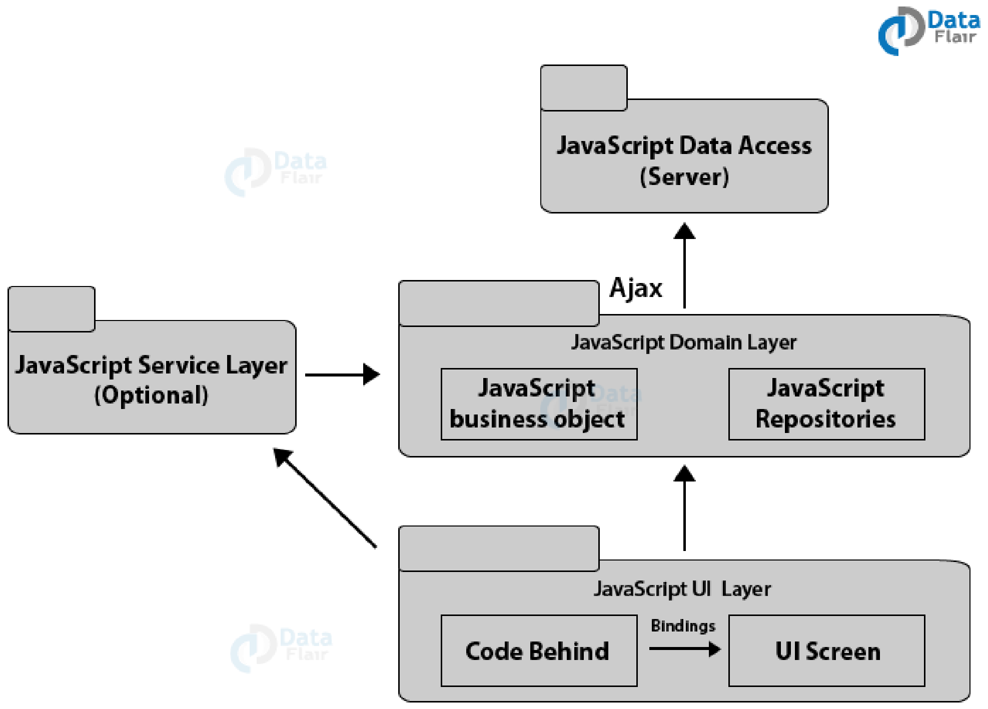
Typical JavaScript Application Architecture
Typically, JavaScript applications use the bottom-up approach, always placing the User Interface (UI) at the center of the development at all times. As shown in the diagram, both the UI and the Server directly link to the code behind. This JavaScript architecture works fine for simple programs, but in the long run, it fails to meet the demands of complex programs. You can use it for most of the websites where you don’t add special frameworks. This architecture is perfectly fine if you want to interact with the application screen, though you might face some trouble in a large-scale application. Hence, it is crucial that you research this architecture before investment and implementation.


Server
Ajax
Code Behind
Controller as JavaScript Object
ViewModel as JavaScript Object
UI Screen
HTML with special attributes
View as JavaScript Object

b)

Framework-based Typical JavaScript Architecture

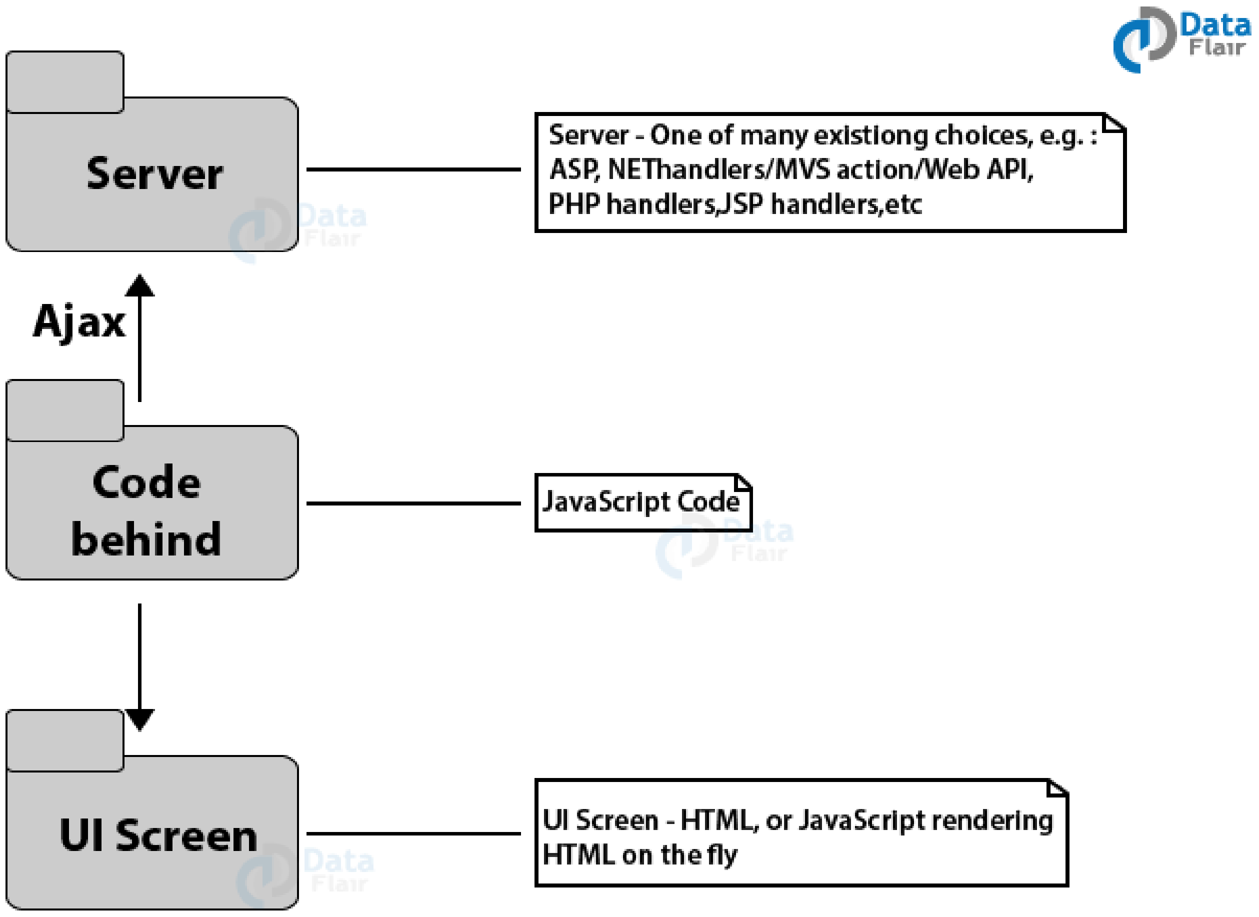
As the demand for a complex architecture of JavaScript arose, Framework-Based Architecture came into use. Even though the main focus is still the UI screen, it is much more advanced than the previous version. It involves detailing the simplistic view of the JavaScript Application Architecture. It is very effective for solving complex problems as it implements either MVC or MVVM pattern to the application. Business and presentation concerns are separate, making it easier to work with. Code behind (Controller and ViewModel) directly link with UI Screen through Bindings and Server through Ajax. UI Screen consists of View and HTML and this architecture has provided the developers with some very advanced features.



c)

Advanced JavaScript Architecture

The prior architecture works fine when you want to create simple web applications. But the problem that arises here is that we are still trying to implement advanced features in the same framework architecture. We are trying to add complexity in a simple framework. These architectures are thus incapable of growing without further maintenance and development costs. The previous versions start having problems that every UI software faces only after the implementation of MVC or MVVM. Controller or View is busy with so many concerns, along with interacting directly with the server and manipulating the view directly or indirectly, that this quickly leads to their failure. These types of problems are detected later in the development, thus we required an architecture that is more focused around domain knowledge. The need for a more mature architecture arose so as to add more features to JavaScript.This is where the Advanced Architecture of JavaScript comes into play. Since the business and UI concerns are separate in this architecture, it adjusts to the needs of the complex applications very easily. It also removes UI as the center of the application, increasing the application's usability. This architecture consists of the following layers for different tasks: In the JavaScript UI Layer, Code-behind(code in the backend) is linked to the UI Screen through Bindings. It can talk to JavaScript Domain Layer either directly or with the help of optional JavaScript Service Layer. JavaScript Domain Layer (JavaScript Business Objects and JavaScript Repositories) is the house of business logic. JavaScript Data Access(Server) can directly talk to the JavaScript Domain Layer by using Ajax calls. Repositories recover JSON objects from the server and map them back to the business objects.


Server
Server - One of many existiong choices, e.g. : ASP, NEThandlers/MVS action/Web API, PHP handlers,JSP handlers,etc
Ajax
Code behind
JavaScript Code
UI Screen
UI Screen - HTML, or JavaScript rendering HTML on the fly
Data Flair

d)

Typical JavaScript Application Architecture
Typically, JavaScript applications use the bottom-up approach, always placing the User Interface (UI) at the center of the development at all times. As shown in the diagram, both the UI and the Server directly link to the code behind.This JavaScript architecture works fine for simple programs, but in the long run, it fails to meet the demands of complex programs. You can use it for most of the websites where you don't add special frameworks. This architecture is perfectly fine if you want to interact with the application screen, though you might face some trouble in a large-scale application. Hence, it is crucial that you research this architecture before investment and implementation.

Question-4

a)

Python code for computing the byte lengths of the text:

Code:

```
from sys import getsizeof
print(getsizeof('中國新年快樂'))
```

Output:

```
86
```

So the byte size length of the string is 86 bytes.

b)

Hexadecimal utf-8 unicode	Chinese letter
4E2D	中
570B	國
65B0	新
5E74	年
5FEB	快
6A02	樂

Hexadecimal - Chinese letter
4E2D
Python code to print this:

```
print(u'\u4E2D\u570B\u65B0\u5E74\u5FEB\u6A02')
```

Output:

```
中國新年快樂
```

Question-5

The XML repersentation of Bach Family is:

```
<ftml>
<people>
<person id="Johann Ambrosius Bach" sex="male" >
</person>
<person id="Maria Elisabeth Lämmerhirt" sex="female">
 </person>
<person id="Johann Sebastian Bach" sex="male">
<wife id= "Anna Magdalena Wilcke" />
<mother id="Maria Elisabeth Lämmerhirt" />
<father id="Johann Ambrosius Bach" />
</person>
<person id="Anna Magdalena Wilcke" sex="female">
<husband id="Johann Sebastian Bach" />
</person>
<person id="Maria Barbara Bach" sex="female">
<husband id="Johann Sebastian Bach" />
</person>
<person id="Wilhelm Friedemann Bach" sex="male" >
<father id="Johann Sebastian Bach" />
<mother id="Maria Barbara Bach" />
```

```
</person>
<person id="Carl Philipp Emanuel Bach" sex="male" >
<father id="Johann Sebastian Bach" />
<mother id="Maria Barbara Bach" />
</person>
<person id="Gottfried Heinrich Bach" sex="male" >
<father id="Johann Sebastian Bach" />
<mother id="Anna Magdalena Wilcke" />
</person>
<person id="Johann Christoph Friedrich Bach" sex="male" >
<father id="Johann Sebastian Bach" />
<mother id="Anna Magdalena Wilcke" />
</person>
<person id="Johann Christian Bach" sex="male" >
<father id="Johann Sebastian Bach" />
<mother id="Anna Magdalena Wilcke" />
</person>
<person id="   Elisabeth Juliane Friederica" sex="female" >
<father id="Johann Sebastian Bach" />
<mother id="Anna Magdalena Wilcke" />
</person>
<person id="   Johanna Carolina" sex="female" >
<father id="Johann Sebastian Bach" />
<mother id="Anna Magdalena Wilcke" />
</person>
<person id="   Regina Susanna" sex="female" >
<father id="Johann Sebastian Bach" />
<mother id="Anna Magdalena Wilcke" />
</person>
</people>
<marriages>
<marriage husband="Dad" wife="Mom" />
<marriage husband="Uncle" wife="Aunt" />
</marriages>
</ftml>
```

After the validation on https://www.xmlvalidation.com/, we get no errors

JSON representation of Bach Family

```
{
   "Bachs": {
      "Johann Sebastian Bach": { "id": 0, "wife": [1,2] , "father": 11, "mother": 12},
      "Anna Magdalena Wilcke": { "id": 1, "husband": 0},
      "Maria Barbara Bach": { "id": 2, "husband": 0 },
      "Carl Philipp Emanuel Bach": { "id": 3,  "father": 0, "mother": 1 },
      "Wilhelm Friedemann Bach": { "id": 4,  "father": 0, "mother": 1  },
      "Gottfried Heinrich Bach": { "id": 5,  "father": 0, "mother": 1  },
      "Johann Christoph Friedrich Bach": { "id": 6,  "father": 0, "mother": 2  },
      "Johann Christian Bach": { "id": 7, "father": 0, "mother": 2  },
      "Elisabeth Juliane Friederica": { "id":8, "father": 0, "mother": 2  },
      "Johanna Carolina": { "id": 9, "father": 0, "mother": 2  },
      "Regina Susanna": { "id": 10,  "father": 0, "mother": 2  },
      "Johann Ambrosius Bach": { "id": 11},
      "Maria Elisabeth Lämmerhirt": { "id": 12}
   }
}
```

After verifying from https://jsonlint.com/ we get

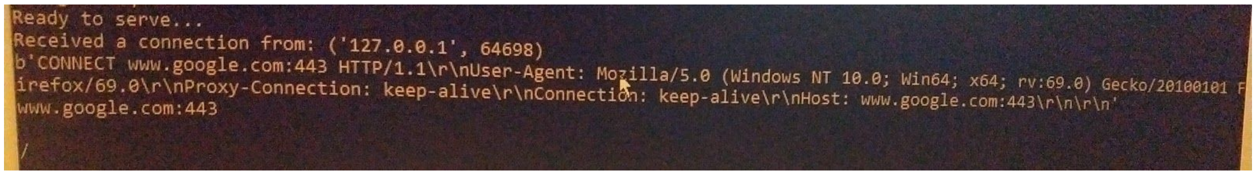
```
{
	"Bachs": {
		"Johann Sebastian Bach": {
			"id": 0,
			"wife": [1, 2],
			"father": 11,
			"mother": 12
		},
		"Anna Magdalena Wilcke": {
			"id": 1,
			"husband": 0
		},
		"Maria Barbara Bach": {
			"id": 2,
			"husband": 0
		},
		"Carl Philipp Emanuel Bach": {
			"id": 3,
			"father": 0,
			"mother": 1
		},
		"Wilhelm Friedemann Bach": {
			"id": 4,
			"father": 0,
			"mother": 1
		},
		"Gottfried Heinrich Bach": {
			"id": 5,
			"father": 0,
			"mother": 1
		},
		"Johann Christoph Friedrich Bach": {
			"id": 6,
			"father": 0,
			"mother": 2
		},
		"Johann Christian Bach": {
			"id": 7,
			"father": 0,
			"mother": 2
		},
		"Elisabeth Juliane Friederica": {
			"id": 8,
			"father": 0,
			"mother": 2
		},
		"Johanna Carolina": {
			"id": 9,
			"father": 0,
			"mother": 2
		},
		"Regina Susanna": {
			"id": 10,
			"father": 0,
			"mother": 2
		},
		"Johann Ambrosius Bach": {
			"id": 11
		},
		"Maria Elisabeth Lämmerhirt": {
			"id": 12
		}
	}
}
```

Question 7

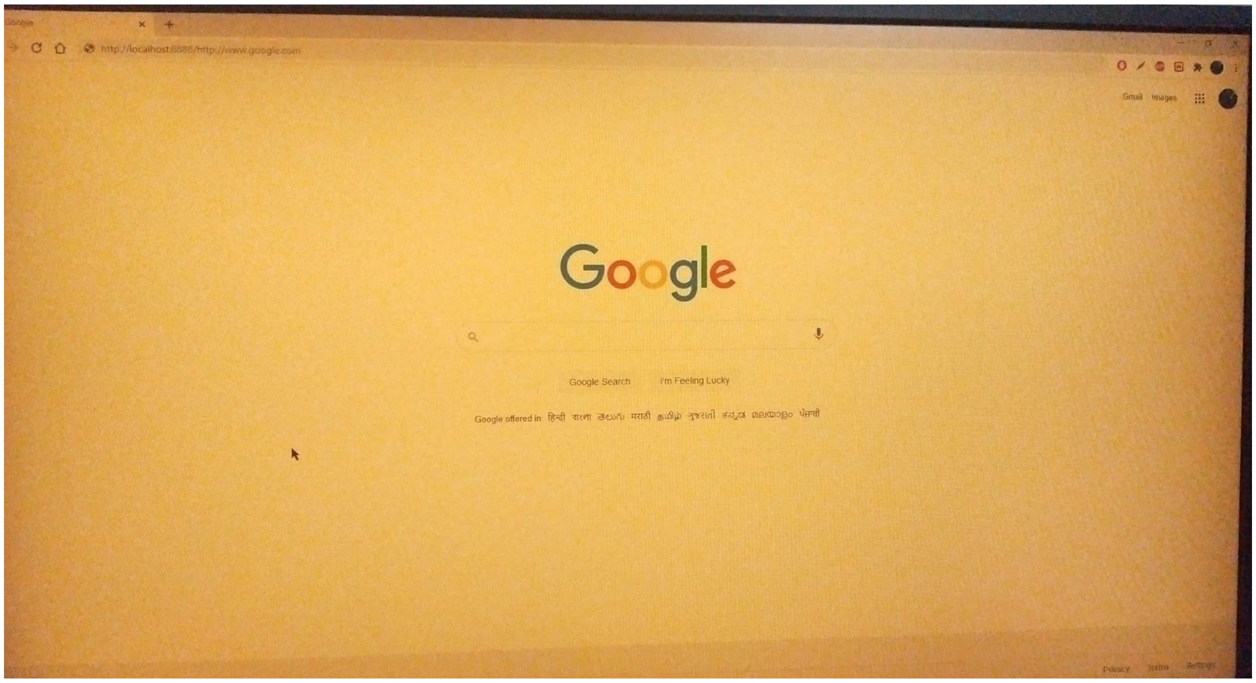
The complete code:

```
from socket import *
import sys

if len(sys.argv) <= 1:
    print('Usage : "python ProxyServer.py server_ip"\n[server_ip : It is the IP Address Of Proxy
Server')
    sys.exit(2)

# Create a server socket, bind it to a port and start listening
tcpSerSock = socket(AF_INET, SOCK_STREAM)
# bind to the port
tcpSerSock.bind(('localhost', 8888))
# start listening
tcpSerSock.listen()  # become a server socket

while 1:
    # Start receiving data from the client
    print('Ready to serve...')
    tcpCliSock, addr = tcpSerSock.accept()
    print('Received a connection from:', addr)
    message = repr(tcpCliSock.recv(1024))
    print(message)
    # Extract the filename from the given message
    print(message.split()[1])
    filename = message.split()[1].partition("/")[2]
    print(filename)
    fileExist = "false"
    filetouse = "/" + filename
    print(filetouse)
    try:
        # Check whether the file exist in the cache
        f = open(filetouse[1:], "r")
        outputdata = f.readlines()
        fileExist = "true"
        # ProxyServer finds a cache hit and generates a response message
        tcpCliSock.send(bytes("HTTP/1.0 200 OK\r\n"))
        tcpCliSock.send(bytes("Content-Type:text/html\r\n"))
        tcpCliSock.send(bytes("Cache hit found hence message generated!!"))
        print('Read from cache')
    # Error handling for file not found in cache
    except IOError:
        if fileExist == "false":
            # Create a socket on the proxy server
            c = socket(AF_INET, SOCK_STREAM)
            hostn = filename.replace("www.", "", 1)
            print(hostn)
            try:
                # Connect to the socket to port 80
                c.bind(('localhost', 80))
                # Create a temporary file on this socket and ask port 80 for the file requested by the
client
                fileobj = c.makefile('r', 0)
                fileobj.write("GET " + "http://" + filename + "HTTP/1.0\n\n")
                # Read the response into buffer
                buffer = c.recv(1024)
                # Create a new file in the cache for the requested file.
                # Also send the response in the buffer to client socket and the corresponding file in
the cache
                tmpFile = open("./" + filename, "wb")
                c.send(buffer)
                tmpFile.write(buffer)
            except:
                print("Illegal request")
        else:
            # HTTP response message for file not found
            tcpCliSock.send(bytes("HTTP/1.1 404 ERROR\n" + "Content-Type: text/html\n" + "\n" +
"<html><body>File Not "
                                                  "found</body></html>\n"))
    # Close the client and the server sockets
    tcpCliSock.close()
    # tcpSerSock.close()
```

The output:



```
Ready to serve...
Received a connection from: ('127.0.0.1', 64698)
b'CONNECT www.google.com:443 HTTP/1.1\r\nUser-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:69.0) Gecko/20100101 F
irefox/69.0\r\nProxy-Connection: keep-alive\r\nConnection: keep-alive\r\nHost: www.google.com:443\r\n\r\n'
www.google.com:443

/
```



```
http://localhost:8888/http://www.google.com
```

Google
Gmail
Images
Google
Google Search
I'm Feeling Lucky
Google offered in हिन्दी বাংলা తెలుగు मराठी தமிழ் ગુજરાતી ಕನ್ನಡ മലയാളം ਪੰਜਾਬੀ
Privacy
Terms
Settings