ANSWERS

A12.8 Client server, REST and thin clients

Networks and the web · A level · OCR H446 1.3.3, AQA 7517 4.9.2.2, Eduqas A500QS 2.5 · about 30 min

What this lesson is about

Peer to peer and client server, websockets, CRUD and REST with JSON, and thin versus thick clients.

- Peer to peer and client server
- The client server model
- CRUD and REST
- JSON and XML
- Thin and thick clients

Questions

6 marks in all

1. Which HTTP method and SQL statement match the CRUD operation Update? [1 mark]

- ☐ **A** PUT and UPDATE
- ☐ **B** POST and INSERT
- ☐ **C** GET and SELECT
- ☐ **D** DELETE and DROP

> **Answer:** A. Create is POST/INSERT, Retrieve is GET/SELECT, Update is PUT/UPDATE, Delete is DELETE/DELETE.

2. How does the websocket protocol differ from ordinary HTTP requests? [1 mark]

- ☐ **A** It keeps a persistent full-duplex connection, so the server can push data to the client at any time
- ☐ **B** It encrypts every message with a digital certificate
- ☐ **C** It only allows the client to send data
- ☐ **D** It uses UDP instead of TCP

> **Answer:** A. After an HTTP upgrade handshake, either side can send over the one open TCP connection without a new request.

3. Why is JSON often preferred to XML for web APIs? Choose all that apply. [1 mark]

Tick every answer that is true.

- ☐ **A** It is more compact
- ☐ **B** It is easier and quicker for a computer to parse
- ☐ **C** It is easier for a human to read
- ☐ **D** It supports schemas that validate structure better

> **Answer:** A, B, C. JSON has no closing tags and maps directly to program objects; schema validation is a traditional strength of XML.

4. A REST API receives DELETE /robots/7, but there is no robot 7. Which status code should it return? [1 mark]

- ☐ **A** 404
- ☐ **B** 200
- ☐ **C** 201
- ☐ **D** 204

> **Answer:** A. 404 Not Found: the resource does not exist. 204 would mean it was deleted with nothing to return.

5. Which are advantages of thin-client computing? Choose all that apply. [1 mark]

Tick every answer that is true.

- ☐ **A** Client devices can be cheap
- ☐ **B** Software and data are managed centrally
- ☐ **C** Clients keep working without a network connection
- ☐ **D** The server carries less load

> **Answer:** A, B. Thin clients rely on the server, so they need the network and put more load on the server.

6. Which is a disadvantage of a peer-to-peer network compared with client-server? [1 mark]

- ☐ **A** There is no central control of security and backups
- ☐ **B** It needs an expensive dedicated server
- ☐ **C** If the server fails, every computer loses the service
- ☐ **D** Peers cannot share files

> **Answer:** A. Each peer manages its own resources; the other options describe client-server networks.

The task: the robot as a REST server

Turn the robot into a tiny REST server. The Client on this mat sends six requests over the radio, one a second. Each is words separated by single spaces: a method, a path, and for some an extra value. The robot has two kinds of resource: `/led`, its LED colour (starting as `off`), and `/moves/<id>`, the moves it has made, each stored as its distance in whole centimetres with ids counting from 1. Handle each request like this: | Request | What to do | Send | |---|---|---| | `GET /led` | nothing | `200 <colour>` | | `PUT /led <colour>` | set the LED to that colour name | `200 <colour>` | | `POST /moves <cm>` | drive forward `<cm>` cm at speed 50, store it under the next id | `201 /moves/<id>` | | `GET /moves/<id>` | nothing | `200 <cm>`, or `404` if there is no such move | | `DELETE /moves/<id>` | remove the move | `204`, or `404` if there is no such move | | any other path | nothing | `404` | After sending each response, print `<request> -> <response>`. Stop after the sixth request, so the output is 6 lines. Check `messages()` every 0.1 s, and build responses from your stored resources rather than typing them.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

colour = "off"
moves = {}
next_id = 1

wait(6)
for sender, text in messages():
    print(text)
```

The hint students can ask for: Split each request into its method, its path and anything after. Keep the resources in variables: the LED colour, and a dictionary of moves by number. Decide the response by the method and whether the path names something that exists, then send it and print the pair.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

colour = "off"
moves = {}
next_id = 1
handled = 0
while handled < 6:
    wait(0.1)
    for sender, text in messages():
        parts = text.split()
        method, path = parts[0], parts[1]
        if path == "/led" and method == "GET":
            response = f"200 {colour}"
        elif path == "/led" and method == "PUT":
            colour = parts[2]
            led(colour)
            response = f"200 {colour}"
        elif path == "/moves" and method == "POST":
            cm = int(parts[2])
            forward(50, distance=cm)
            moves[next_id] = cm
```

```
            response = f"201 /moves/{next_id}"
            next_id = next_id + 1
        elif path.startswith("/moves/"):
            number = int(path.split("/")[2])
            if number not in moves:
                response = "404"
            elif method == "GET":
                response = f"200 {moves[number]}"
            elif method == "DELETE":
                del moves[number]
                response = "204"
            else:
                response = "405"
        else:
            response = "404"
        send(response)
        print(text, "->", response)
        handled = handled + 1
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.