Attempt 1 (9/2/2025) 21.5/57.5 = 37.4 %

Web Architecture Quiz – React + Flask + HTTP + APIs

Quiz Questions

Section 1 – Concepts (1/1)

1. What is the role of React in your project?
   a) Rendering HTML templates from Flask
   b) Handling UI and forms in the browser
   c) Storing data in MongoDB
   d) Training ML models

2. What is the role of Flask in your project? (1/1)
   a) Rendering and styling your table UI
   b) Acting as a JSON API layer between React, the database, and Notion
   c) Managing Wi-Fi connections
   d) Replacing MongoDB

3. Why is WTForms not needed in your architecture? (1/1)

React handles forms, Flask is not serving any files, it just provides apis

4. True or False: In your system, React and Flask both render the UI. (1/1)

False

Section 2 – HTTP + Networking

5. What is an HTTP request? (1/1)
   a) A JavaScript function
   b) A letter-like message sent from a client to a server, structured with method, headers, and optional body
   c) A way to style web pages
   d) A database query

-2/ 6. When React sends a fetch() call to Flask, what network layers carry that request? Order them: (0/2)

3 - TCP
1 - HTTP
2 - IP
4 - Physical medium (Wi-Fi/Ethernet)

-2

-2

7. What does it mean when we say "HTTP over TCP/IP"? (0/2)

A request is initiated from the client side to a server (HTTP). The request uses IP addresses to know where to send the request to & where to respond.

-2 ??

8. What role does the browser play in this context?

(0/2) TCP restructures the http request to be compatible with wifi

Section 3 – APIs

9. What is an API in the context of web development?

a) A Python library for ML (0/1)

b) A database table schema

c) A set of defined HTTP endpoints with request/response formats

d) A styling framework

(2/2) 10. How does Flask serve as both an API provider and an API consumer in your project?

Provides endpoints for React to request stat calcs, submit plan data, check for connectivity issues. Consumes api requests when requesting database internally & Notion API when responding to React requests

(2/3) 11. Fill in the blanks:

- React sends JSON to Flask → Flask receives the data, validates it to the DB/Notion/ML, then sends JSON back to React.

-1

Section 4 – Application Flow

(2.5/2.5) 12. Put the following steps in order for what happens when you enter a task and hit "Submit":

3 - Flask validates JSON

1 - React makes POST /api/tasks

2 - Browser transmits HTTP request via TCP/IP

4 - Flask responds with confirmation JSON

5 - React updates UI with success message

(1/1) 13. True or False: The physical medium carrying HTTP requests could be Wi-Fi, Ethernet cables, or fiber optics.

True

(1/2) 14. Why is it considered best practice today to let React handle the UI and Flask only handle APIs, instead of Flask rendering HTML directly?

-1

- React allows for dynamic web pages while Flask serves relatively static html files. Also react is entirely built using only 1 HTML page & uses Javascript to render things as though there are multiple html pages

-6

Extended Response Questions (HTTP, TCP, IP)

(1/6) 15. Explain how TCP differs from IP.
- Include what job each one does in delivering data between computers.
- Why do we need both?

-1 (1/2) IP: internet Protocol, provides the address?
-2 (0/2) TCP: transmission Control Protocol, restructures data to be compatible with wifi?

-2 (0/2)

-5

(0/5) 16. Why is HTTP called a "stateless protocol"?
- What does statelessness mean in practice for a web application?
- How do things like cookies or tokens add "state" back in when needed?

-5

(2/4) 17. Draw an analogy between HTTP/TCP/IP and the real world.
- Identify what corresponds to the roads, the vehicles, the letter, and the language inside the letter.

(2/2) HTTP - letter + language?
(0/1) IP - vehicle
(0/1) TCP - road

-2

(0/6) 18. What are the roles of "port numbers" in TCP/IP communication?
- Why might Flask default to port 5000 or 5001?
- How does the browser know where to deliver the HTTP request on a given machine?

???

-6

-18

Article Questions: Best Practices for Flask API Development

URL: https://auth0.com/blog/best-practices-for-flask-api-development/

(1/2) 19. What are the recommended conventions for naming API endpoints in a Flask application?

- all lowercase
- use forward slashes for hierarchy '/'
- use hyphens for descriptions
- use nouns in plural form to represent resources
- −1 • use verbs for actions — don't use verbs

(0/2) 20. How should plural vs. singular nouns be used in RESTful API route naming?

-2

(0/2) 21. Why is it important to use nouns instead of verbs in API paths (e.g., /users vs. /getUsers)?

-2

(0/2) 22. What are the best practices for choosing between query parameters and request body data in Flask APIs?

-2

(2/2) 23. Why should consistent naming patterns (e.g., /tasks/{id} instead of mixing /taskById/{id} and /tasks/{id}) be enforced across endpoints?

- to lower tech debt, not make things needlessly confusing
- improves reusability, readability

(1/2) 24. How does proper naming of error responses (e.g., 404 Not Found vs. a vague error) improve API usability?

- helps the developer pinpoint issues & debug their code
- lets the user know what went wrong

−1

(0/2) 25. What are some strategies for keeping API route names intuitive while supporting complex operations (e.g., filtering, sorting)?

-2

-10

Attempt 2 (9/4/2025)

45.5/57.5 = 79%

-12

Good Enough!

Web Architecture Quiz – React + Flask + HTTP + APIs

Quiz Questions

Section 1 – Concepts (1 point)

1. What is the role of React in your project?
   a) Rendering HTML templates from Flask
   b) Handling UI and forms in the browser
   c) Storing data in MongoDB
   d) Training ML models

2. What is the role of Flask in your project? (1 point)
   a) Rendering and styling your table UI
   b) Acting as a JSON API layer between React, the database, and Notion
   c) Managing Wi-Fi connections
   d) Replacing MongoDB

3. Why is WTForms not needed in your architecture? (1 point)

React handles the frontend & UI related stuff

4. True or False: In your system, React and Flask both render the UI. (1 Point)

F

Section 2 – HTTP + Networking

5. What is an HTTP request? (1 point)
   a) A JavaScript function
   b) A letter-like message sent from a client to a server, structured with method, headers, and optional body
   c) A way to style web pages
   d) A database query

6. When React sends a fetch() call to Flask, what network layers carry that request? Order them: (2 points)

2 - TCP
1 - HTTP
3 - IP
4 - Physical medium (Wi-Fi/Ethernet)

7/7

7. What does it mean when we say "HTTP over TCP/IP"? (2 points)

HTTP is the message being carried above the TCP/IP layers.

8. What role does the browser play in this context? (2 points)

- bridges gap between javascript & network

Section 3 – APIs

9. What is an API in the context of web development? (1 point)

a) A Python library for ML
b) A database table schema
c) A set of defined HTTP endpoints with request/response formats
d) A styling framework

10. How does Flask serve as both an API provider and an API consumer in your project? (2 points)

provide: react, consume: notion, DB

11. Fill in the blanks: (3 points)

- React sends JSON to Flask → Flask _____ the data, _____ it to the DB/Notion/ML, then _____ JSON back to React.

verifies sends responds

Section 4 – Application Flow

12. Put the following steps in order for what happens when you enter a task and hit "Submit": (2.5 points)

3 - Flask validates JSON
1 - React makes POST /api/tasks
2 - Browser transmits HTTP request via TCP/IP
4 - Flask responds with confirmation JSON
5 - React updates UI with success message

13. True or False: The physical medium carrying HTTP requests could be Wi-Fi, Ethernet cables, or fiber optics. (1 point)

T

14. Why is it considered best practice today to let React handle the UI and Flask only handle APIs, instead of Flask rendering HTML directly? (2 points)

- React allows for a more interactive experience using Javascript close to the browser
- Flask has a limited capacity for serving dynamic web pages
- Flask is good for managing connections with other apps

15.5 /15.5

Extended Response Questions (HTTP, TCP, IP)

15. Explain how TCP differs from IP. (6 points)
- Include what job each one does in delivering data between computers.
- Why do we need both?

TCP - divies up, orders messages, ensures they arrive in good quality & order, reassembly

IP - gives addresses to send & destination PCs & determines the route

IP ensures messages have a path to move, TCP manages the section & ensures it arrives in good quality

16. Why is HTTP called a "stateless protocol"? (5 points) -1
- What does statelessness mean in practice for a web application?
- How do things like cookies or tokens add "state" back in when needed?

- ~~HTTP~~ is stateless b/c it's message is never permanently stored in the process of manging a session by client or server
- previous actions are not stored, harder to track history, making the same request multiple times is not a problem

X- Cookies attach & store ids associated with the request & can be retrieved later on

17. Draw an analogy between HTTP/TCP/IP and the real world. (4 points)
- ~~Identify what corresponds to the roads, the vehicles, the letter, and the language inside the letter.~~

HTTP = message (big boxes)

TCP - divies up box into multiple volumes, then ensures each volume arrives & in the right order, reassembles back once they arrive

IP - address of sender + receiver + finds best shipping route

Physical medium - Vehicle carrying package, roads used

18. What are the roles of "port numbers" in TCP/IP communication? (6 points)
- Why might Flask default to port 5000 or 5001?
- How does the browser know where to deliver the HTTP request on a given machine?

-2 .??

-1 . default to 5000 or 5001 b/c it's not commonly used

. the port numbers?

-2

-6

15/21

Article Questions: Best Practices for Flask API Development

URL: https://auth0.com/blog/best-practices-for-flask-api-development/

19. What are the recommended conventions for naming API endpoints in a Flask application? (2 points)

- don't use verbs
- use hyphens for detailed explanations
- use forward slashes for hierarchy

20. How should plural vs. singular nouns be used in RESTful API route naming? (2 points)

-2

21. Why is it important to use nouns instead of verbs in API paths (e.g., /users vs. /getUsers)? (2 points)

- you can attach more than GET HTTP requests to an api endpoint
- can make a singular endpoint more versatile in a way that makes sense, ie GET users, POST users, DELETE users, makes this api endpoint more intuitive

22. What are the best practices for choosing between query parameters and request body data in Flask APIs? (2 points)

-2

23. Why should consistent naming patterns (e.g., /tasks/{id} instead of mixing /taskById/{id} and /tasks/{id}) be enforced across endpoints? (2 points)

- make things more readable, reusable, maintainability, scalability easier to build on
- APIs are used by other people so want to minimize sources of confusion & complexity

24. How does proper naming of error responses (e.g., 404 Not Found vs. a vague error) improve API usability? (2 points)

- helps developer debug easier
- helps with maintainability & reusability

25. What are some strategies for keeping API route names intuitive while supporting complex operations (e.g., filtering, sorting)? (2 points)

-2

-6

8/14

Answer Key

Section 1 – Concepts

1. b
2. b
3. WTForms is unnecessary since React handles forms.
4. False

Section 2 – HTTP + Networking

5. b
6. Physical → IP → TCP → HTTP
7. HTTP rides on top of TCP/IP transport.
8. Browser = runtime + messenger for React.

Section 3 – APIs

9. c
10. Provider to React, consumer of Notion/MongoDB.
11. Validate, forward, return.

Section 4 – Application Flow

12. React POST → Browser TCP/IP → Flask validate → Flask respond → React update UI
13. True
14. Separation of concerns: React UI, Flask API = modern best practice.

Section 5 – Extended Response

15. Explain how TCP differs from IP.

- IP (Internet Protocol): Handles addressing and routing. It makes sure data packets get from the source computer to the destination computer (like a postal system ensuring the envelope arrives at the right house).

- TCP (Transmission Control Protocol): Ensures reliable, ordered delivery of data between applications. It breaks large messages into packets, numbers them, ensures none are lost, and reassembles them at the destination.

- Why we need both: IP gets the data to the right place, but it doesn't guarantee completeness or order. TCP adds reliability and sequencing so applications can trust the data they receive.

16. Why is HTTP called a “stateless protocol”?

- Statelessness: Each HTTP request is independent; the server doesn’t remember anything about previous requests. For example, visiting a website homepage and then clicking a link sends two separate requests, and the server treats them as unrelated.

- In practice: Without extra mechanisms, the server can’t “remember” if a user is logged in or what’s in their shopping cart.

- How cookies/tokens add state: Cookies (stored in the browser) or tokens (like JWTs) are sent with each request. These identifiers let the server link requests together and simulate “state” (e.g., recognizing you’re logged in).

17. Draw an analogy between HTTP/TCP/IP and the real world.

- IP = Roads: The infrastructure that allows vehicles (packets) to travel between locations (computers).

- TCP = Vehicles: The trucks or delivery vans that carry letters reliably, making sure none are lost or delivered out of order.

- HTTP = Letter contents: The actual message written inside the letter (like a request to fetch a webpage).

- Envelope address = IP address/port: Ensures the letter reaches the correct house (computer) and the right person (application).

- Language in the letter = HTTP format (headers, body): Both sender and receiver must understand it to communicate.

18. What are the roles of “port numbers” in TCP/IP communication?

- Role of ports: Ports identify which specific application/service on a computer should receive the data. Multiple services can run on the same machine, and ports act like “apartment numbers” inside a building.

- Why Flask defaults to 5000/5001: These are conventionally unused ports chosen to avoid conflict with reserved ports like 80 (HTTP) or 443 (HTTPS). They give developers a safe default for local testing.

- How browsers know where to deliver: The browser includes the port number in the request (e.g., `http://localhost:5000`). The operating system uses the port to hand the request to the correct application (like Flask).

Section 6 – Article: API Naming Best Practices

19. What are the recommended conventions for naming API endpoints in a Flask application?

→ Use lowercase, hyphen-separated nouns (kebab-case), keep them descriptive, and avoid verbs in the path.

Example: `/api/v1/tasks` ✅ instead of `/api/v1/getTasks` ❌

20. How should plural vs. singular nouns be used in RESTful API route naming?

→ Use plural nouns for collections (`/users`, `/tasks`) and singular nouns with IDs for specific resources (`/users/123`).

21. Why is it important to use nouns instead of verbs in API paths (e.g., `/users` vs. `/getUsers`)?

→ Because the HTTP method already defines the action (`GET`, `POST`, `PUT`, `DELETE`). Verbs in the path duplicate that meaning and break RESTful conventions.

22. What are the best practices for choosing between query parameters and request body data in Flask APIs?

→ Use query parameters for filtering, sorting, or pagination (non-sensitive, short info).

→ Use the request body for creating or updating resources (larger, structured data).

23. Why should consistent naming patterns (e.g., `/tasks/{id}` instead of mixing `/taskById/{id}` and `/tasks/{id}`) be enforced across endpoints?

→ Consistency makes the API predictable, easier to use, and less error-prone, reducing the learning curve for clients.

24. How does proper naming of error responses (e.g., `404 Not Found` vs. a vague error) improve API usability?

→ Clear, standardized error responses help clients debug faster, improve developer experience, and make APIs self-explanatory.

25. What are some strategies for keeping API route names intuitive while supporting complex operations (e.g., filtering, sorting)?

→ Use query parameters (`/tasks?status=completed&sort=due_date`) instead of cluttering route names.

→ Keep route paths focused on resources rather than actions or operations.