CERTLABZ CERTIFICATION BOARD

Certified Web3 Application Engineer (CW3E)

Ten Sample Questions

These ten questions are written at the difficulty of the live exam. Six carry a figure, a data table, a configuration or a code listing; four are plain scenario questions, as on the live paper. Every one is a scenario a working Web3 application engineer would meet, and none of them appears in the live question bank.

The live exam is multistage adaptive: how you score on one stage selects the difficulty of the next, so no two candidates sit the same paper. It is a performance-based exam: each form can carry up to 5 to 12 performance-based questions alongside the multiple-choice questions, which means you build, sequence or complete something rather than only choosing from a list. Answers and full explanations follow each question here; the live exam does not disclose item-level answers.

Question 1 — Domain 1: Web3 Foundations — DATA

Emma Wilson matches each dApp piece to its job.

Piece	Job
Wallet	(to choose)
Provider	(to choose)
Smart contract	(to choose)
Frontend	(to choose)

Table 1.1 dApp pieces.

Which set of jobs is correct?

A. Talks to the chain via RPC, holds keys and signs, renders the interface, and holds the on-chain logic, in the order shown here.

B. Holds keys and signs, talks to the chain via RPC, holds the on-chain logic, and renders the user interface, in that order.

C. Renders the interface, holds the on-chain logic, holds keys and signs, and talks to the chain via RPC, in the order shown.

D. Holds the on-chain logic, renders the interface, talks to the chain via RPC, and holds keys and signs, in the order shown.

Correct answer: B
The wallet holds keys and signs, the provider talks to the chain over RPC, the smart contract holds the on-chain logic, and the frontend renders the interface. Only option B pairs each piece with its job.

Question 2

Domain 2: Frontend and Wallet Integration

CONFIG

James Carter lists the steps to connect a wallet and send a transaction, out of order.

```
A. Send the transaction and await the receipt
B. Request the user accounts from the wallet
C. Ask the user to sign the transaction
D. Detect and, if needed, switch the network
```

Listing 2.1 Steps to order.

What is the correct order?

A. A, then C, then B, then D, sending and signing the transaction before the accounts are requested or the network is set.

B. B, then D, then C, then A, requesting accounts, ensuring the right network, asking to sign, then sending and awaiting the receipt.

C. D, then A, then B, then C, switching the network and sending before requesting accounts or asking the user to sign at all.

D. C, then B, then A, then D, asking to sign before accounts are known and switching the network only after sending.

Correct answer: B
The flow is: request the user accounts, detect and switch to the right network if needed, ask the user to sign, then send the transaction and await the receipt. The other orders send or sign before the accounts or network are established.

Question 3

Domain 3: Smart Contract Interaction

CONFIG

Olivia Davis reviews two contract interactions.

```
getBalance(addr)   : reads a value, no state change
transfer(to, amt)  : changes state, moves tokens
```

Listing 3.1 Two interactions.

Which interaction needs a signed, gas-paying transaction, and why?

A. Both need a signed transaction, because every interaction with a contract changes state and therefore always costs gas.

B. Only transfer, because it changes state, while getBalance is a read that can be served as a call without a transaction.

C. Only getBalance, because reading a value is what changes state, and transfer is a free read that never costs any gas.

(D) Neither needs a transaction, because a frontend can change on-chain state directly without signing or paying any gas.

Correct answer: B
transfer changes state, so it requires a signed transaction and gas, while getBalance only reads a value and can be served as a call with no transaction or gas. Reads do not change state, and a frontend cannot change on-chain state without a signed transaction.

Question 4 — Domain 4: Storage, Identity and Infrastructure — DATA

Liam Anderson matches each need to the right infrastructure choice.

Need	Choice
Store large files off chain	(to choose)
Query historical events fast	(to choose)
Talk to the chain	(to choose)
Human-readable address	(to choose)

Table 4.1 Needs and choices.

Which set of choices is correct?

(A) An indexer, a naming service, an RPC node and decentralized storage, so large files are stored by the naming service instead.

(B) Decentralized storage, an indexer, an RPC node or provider, and a naming service, matching each need to its right tool.

(C) An RPC node, decentralized storage, a naming service and an indexer, so the chain is queried through the naming service.

(D) A naming service, an RPC node, an indexer and decentralized storage, so historical events are served by the naming service.

Correct answer: B
Large files go in decentralized storage, fast historical event queries use an indexer, talking to the chain uses an RPC node or provider, and a human-readable address uses a naming service. Only option B maps each need to the right choice.

Question 5 — Domain 5: Security and UX — CONFIG

Ava Thompson finds this in a frontend codebase.

```
const PRIVATE_KEY = "0xabc...";  // in frontend code
wallet = new Wallet(PRIVATE_KEY);
```

Listing 5.1 A frontend secret.

What is the problem, and the correct approach?

A. No problem, because embedding a private key in frontend code is a normal way to let the app sign on the user behalf.

B. A private key must never live in frontend code; signing belongs in the user wallet, which holds the key and signs locally.

C. The only problem is the variable name, so renaming PRIVATE_KEY to something less obvious makes this code safe to ship.

D. The only problem is the key length, so using a longer key string in the frontend fully resolves the security concern.

Correct answer: B
A private key in frontend code is exposed to everyone who loads the page and can be stolen instantly. Signing must be delegated to the user wallet, which holds the key and signs locally. Renaming the variable or lengthening the key does nothing to fix the exposure.

Question 6

Domain 5: Security and UX

DATA

A dApp shows a pending transaction. The table lists UX states.

State	Good practice
Transaction submitted	Show pending with the hash
Transaction confirmed	Show success and update UI
Transaction failed	Show a clear error
Wallet rejected	Explain, allow retry

Table 6.1 Transaction UX states.

Why does handling every one of these states matter?

A. It does not matter, because a dApp can safely assume every submitted transaction confirms and never needs any error handling.

B. Because transactions can be slow, fail or be rejected, and clear feedback for each state prevents confusion and repeated sends.

C. It matters only for the confirmed state, since the other states never occur once a transaction has been submitted to the chain.

D. It matters only to make the interface colorful, because the transaction states have no effect on how users actually behave.

Correct answer: B
On-chain transactions can be slow, fail or be rejected by the user, so a dApp must give clear feedback for each state to prevent confusion and duplicate submissions. Assuming every transaction confirms, ignoring non-confirmed states, or treating this as mere decoration all lead to a poor and risky experience.

Question 7

Domain 1: Web3 Foundations

A developer expects a dApp frontend to change on-chain state directly the way a traditional app writes to its own server. How do you correct this?

A. Agree, because a frontend can write to the chain directly whenever it wants, exactly like writing to its own backend server.

B. Explain that changing state requires a signed transaction from the user wallet, which the frontend requests but cannot bypass.

C. Explain that a frontend changes state by editing the block directly, so no wallet or signature is involved in a write at all.

D. Agree, because raising the gas limit lets the frontend write to the chain without any user signature being required.

Correct answer: B
On-chain state changes only through a signed transaction from the user wallet, which the frontend requests but cannot perform on the user behalf without a signature. A frontend cannot edit blocks directly, and gas limits do not remove the signature requirement.

Question 8 Domain 6: Deployment and Operations

Before pointing a dApp at mainnet, why deploy and exercise it on a testnet first?

A. There is no reason, because a dApp that compiles is guaranteed to work identically on mainnet with real funds at stake.

B. A testnet lets the team validate contract interactions and UX with no real funds at risk before committing to mainnet.

C. A testnet is only for marketing screenshots, so the team should skip it and deploy straight to mainnet to save time.

D. A testnet permanently replaces mainnet, so once the dApp works on a testnet it never needs a mainnet deployment at all.

Correct answer: B
A testnet lets the team validate contract interactions, gas behavior and UX with no real funds at risk before mainnet, catching problems cheaply. Compiling does not guarantee correct behavior, a testnet is more than screenshots, and it does not replace the eventual mainnet deployment.

Question 9 Domain 4: Storage, Identity and Infrastructure

A dApp needs to display a large media file attached to an on-chain record. Why store the file off chain and keep only a reference on chain?

A. There is no benefit, because storing large media directly on chain is cheap and the recommended default for any dApp.

B. On-chain storage is expensive, so large files go in decentralized storage and the chain holds a compact reference or hash.

C. Because on-chain storage deletes files after each block, so media must live off chain to survive beyond a single block.

D. Because the chain cannot store any reference at all, so both the file and its reference must live entirely off chain.

Correct answer: B
Writing large data on chain is very expensive, so the file is kept in decentralized storage and the chain holds only a compact reference or content hash. On-chain storage is not cheap for large media, blocks do not delete data, and the chain can hold a reference.

Question 10 Domain 2: Frontend and Wallet Integration

A dApp asks users to approve an unlimited token allowance to a contract for convenience. What is the safer default, and why?

A. Unlimited allowances are always safest, because a larger approval reduces prompts and cannot be misused by a contract.

B. Request only the allowance needed for the action, since an unlimited approval lets a compromised contract drain the token.

C. Ask users to share their seed phrase instead of approving, because that is a faster way to grant spending permission.

D. Approve unlimited for every token at once on connect, because pre-approving everything improves the user experience most.

Correct answer: B
The safer default is to request only the allowance the action needs, because an unlimited approval lets a buggy or malicious contract drain the whole token balance later. Unlimited approvals are a known drain vector, seed phrases must never be shared, and pre-approving everything maximizes risk.

Published by the CertLabz Certification Board, a division of CertLabz LLC. Companion to objectives document CW3E-001. https://certlabz.com/certifications/certified-web3-application-engineer.html