UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Laboratorio 5 – JWT e Programmazione Web in Sicurezza

Corso di ***Sicurezza dei Dati***

Prof. Esposito Christian

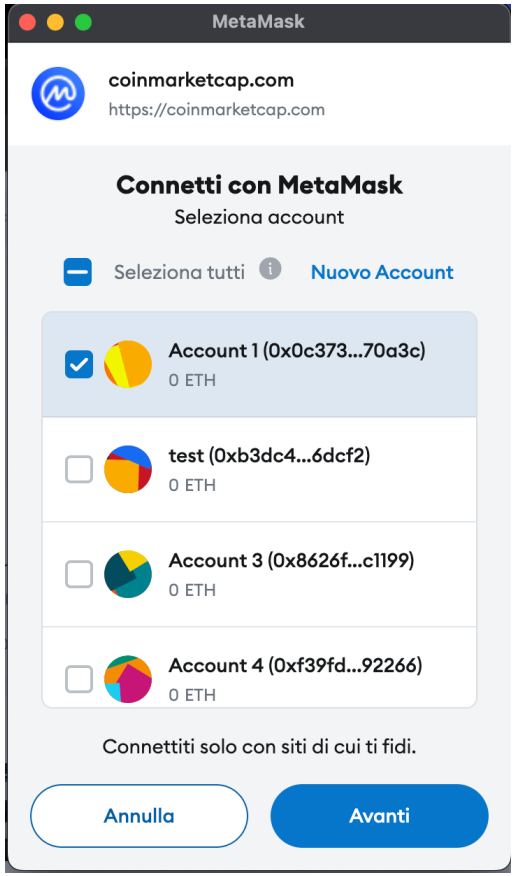
::.. Autenticazione Web

L'autenticazione web è il processo mediante il quale un sistema verifica l'identità di un utente o di un dispositivo che tenta di accedere a risorse protette online. È un passaggio fondamentale per garantire la sicurezza delle informazioni e delle operazioni su internet, poiché permette di assicurarsi che solo utenti autorizzati possano accedere a determinate funzionalità o dati.

Metodi tradizionali basati su username e password non sono più considerati sicuri, e spesso vengono sostituiti da meccanismi più complessi come autenticazione a due fattori (2FA)

::.. Autenticazione DApp

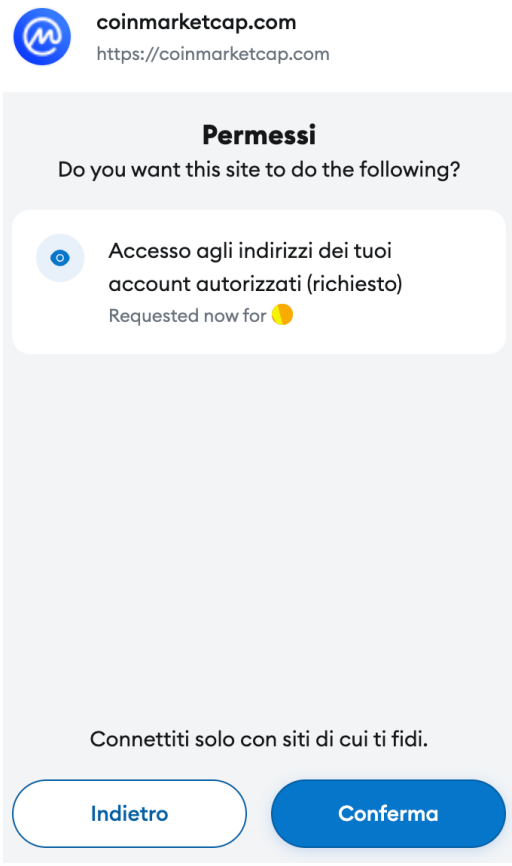
L'autenticazione nelle DApp (Decentralized Applications) si differenzia notevolmente dall'autenticazione tradizionale in quanto sfrutta portafogli digitali (wallet) e chiavi crittografiche anziché password o account centralizzati. La sicurezza e l'identità dell'utente nelle DApp si basano su chiavi crittografiche, solitamente private, che sono associate agli indirizzi univoci dei wallet usati per interagire con la blockchain.



Standard come Sign-in with Ethereum e Sign-in with Wallet vengono utilizzati per la verifica degli indirizzi associati al Wallet.

::.. Autenticazione DApp

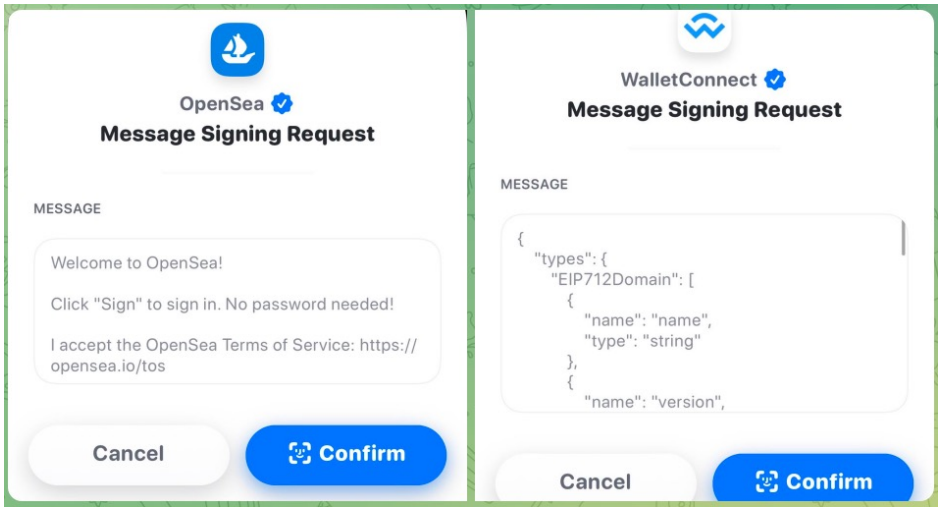
L'autenticazione nelle DApp (Decentralized Applications) si differenzia notevolmente dall'autenticazione tradizionale in quanto sfrutta portafogli digitali (wallet) e chiavi crittografiche anziché password o account centralizzati. La sicurezza e l'identità dell'utente nelle DApp si basano su chiavi crittografiche, solitamente private, che sono associate agli indirizzi univoci dei wallet usati per interagire con la blockchain.



Standard come Sign-in with Ethereum e Sign-in with Wallet vengono utilizzati per la verifica degli indirizzi associati al Wallet.

::.. Sign-In with Ethereum / Wallet

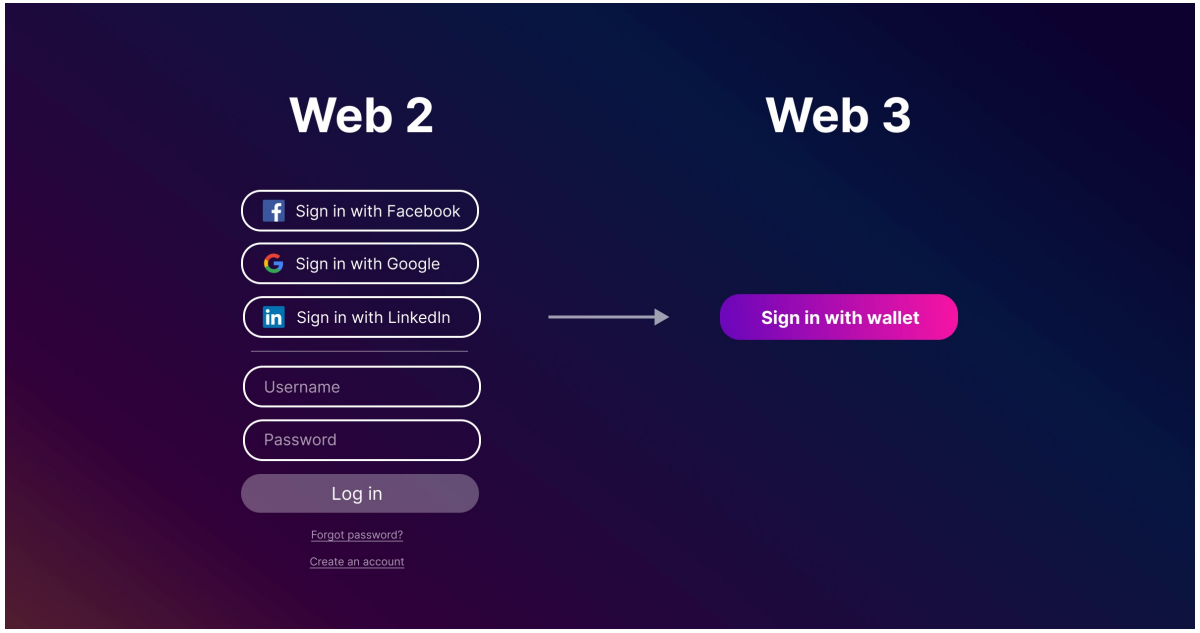
Durante il processo di accesso, l'utente riceverà una richiesta per firmare un messaggio utilizzando il proprio wallet digitale. Questa *challenge* inviata dal server è univoca per ogni tentativo di login. Una volta firmato, il client invia il messaggio firmato, insieme alle informazioni rilevanti contenute nel messaggio (come un identificativo temporaneo o un timestamp), al server. Il server verifica quindi la firma per confermare che proviene effettivamente dal wallet associato all'account dell'utente.



Se la firma risulta valida, l'accesso viene autorizzato, permettendo così all'utente di accedere alla propria area riservata in sicurezza.

::.. Pratical Example

Testiamo l’autenticazione con Ethereum Wallet creando una pagina web dove chiediamo all’utente di autenticarsi utilizzando il proprio wallet.

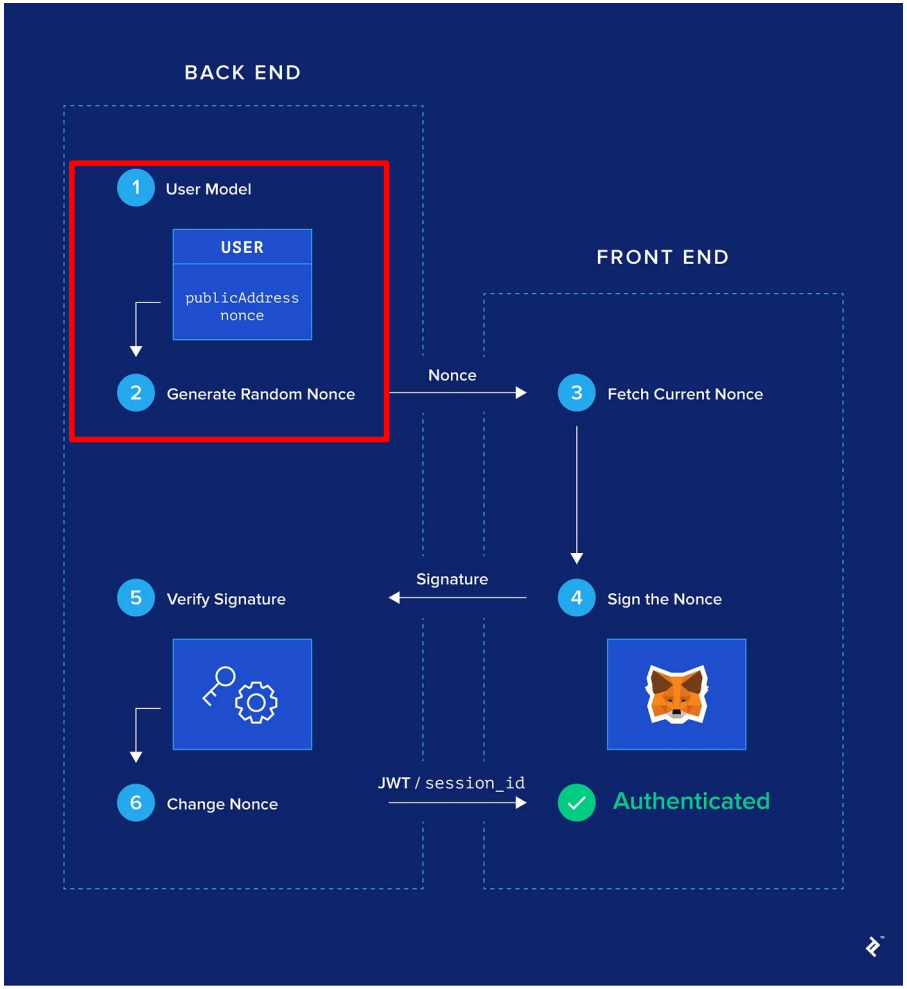


::.. Nonce Generation

L'autenticazione basata su Wallet prevede che l'utente sia identificato all'interno del sistema tramite il suo indirizzo pubblico.

Il primo step è quello di creare un endpoint per la generazione dei nonce, che dovranno essere gestiti da parte del backend.

I nonce sono necessari per evitare attacchi di replay.

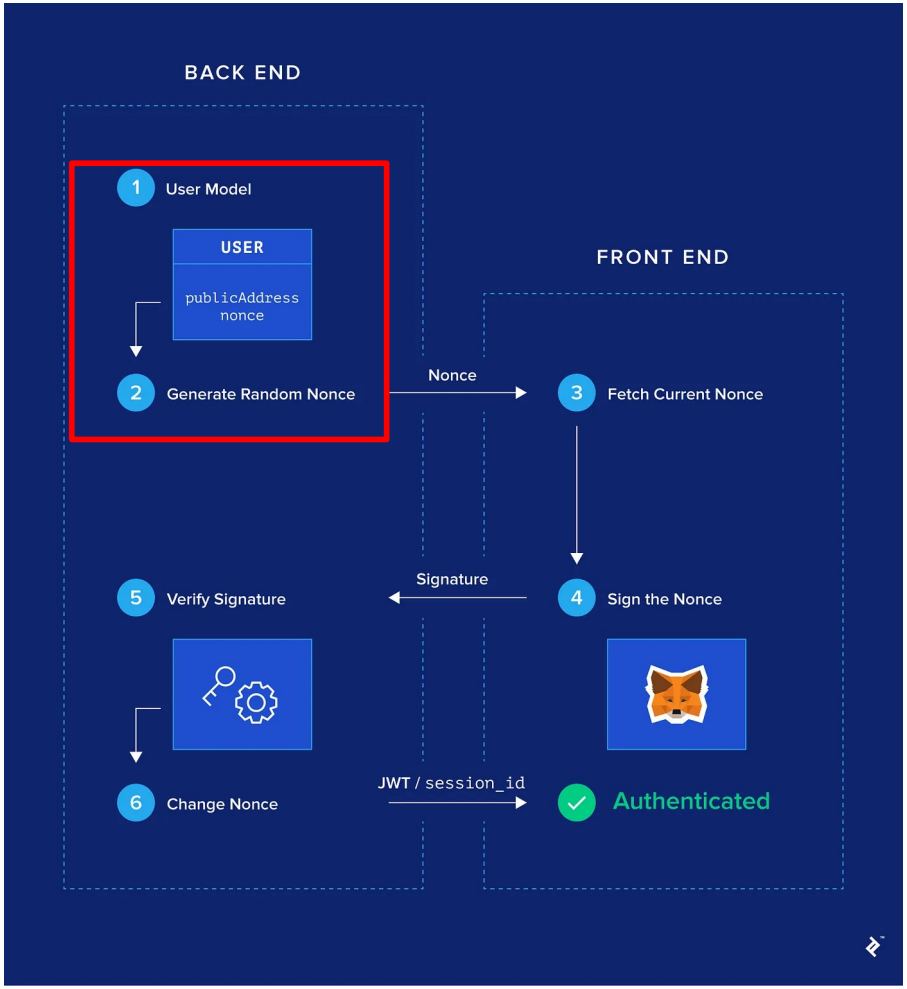


::.. Nonce Generation

Una volta inizializzato il server mediate Express.js è necessario andare a definire i diversi endpoint validi per le richieste.

Il primo step è quello di creare un endpoint per la generazione dei nonce, che dovranno essere gestiti da parte del backend.

```
// Endpoint per generare un nonce univoco per ogni richiesta
app.get("/getNonce", (req, res) => {
    const nonce = `Richiesta autenticata ${Date.now()}`;
    released_nonce.push(nonce);
    res.json({ nonce });
});
```



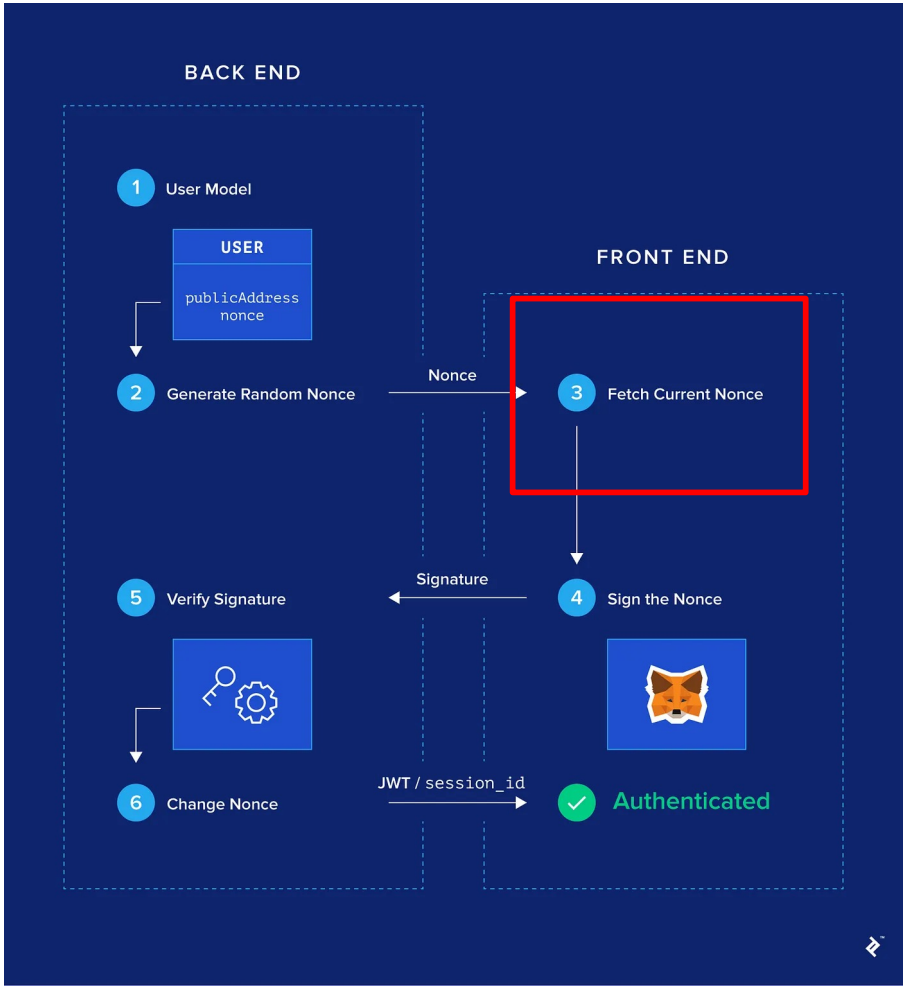
::.. Fetch Nonce

Una volta definita la funzione del back-end (Express) è necessario realizzare la funzione di fetch nel front-end (Javascript).

E’ possibile verificare la presenza del Wallet Metamask attraverso l’oggetto *window.ethereum*

```
// Verifica che MetaMask sia disponibile
if (typeof window.ethereum !== "undefined") {
    console.log("MetaMask è disponibile!");
}
```

```
// Recupera un nonce dal server
const nonceResponse = await fetch("http://localhost:3000/getNonce");
const { nonce } = await nonceResponse.json();
```



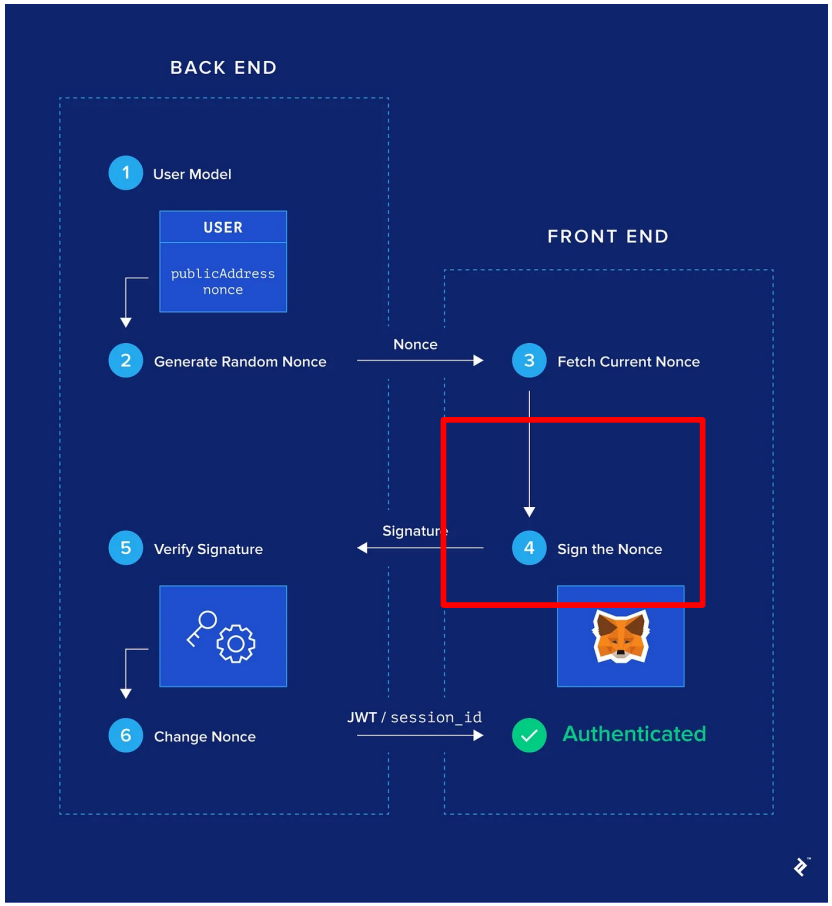
::.. Sign Nonce

A questo punto si può usare il metodo request per effettuare una richiesta di inidirzzo «eth_requestAccounts» ed una richiesta di «personal_sign», passando come parametro l'indirizzo e il nonce.

```
// Richiede l'indirizzo dell'account dall'utente
const accounts = await ethereum.request({ method: "eth_requestAccounts" });
const account = accounts[0];

// L'utente firma il nonce
const signature = await ethereum.request({
    method: "personal_sign",
    params: [nonce, account],
});

// Invia il nonce, la firma e l'account al server per la verifica
const response = await fetch("http://localhost:3000/verify", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ account, signature, nonce }),
});
```



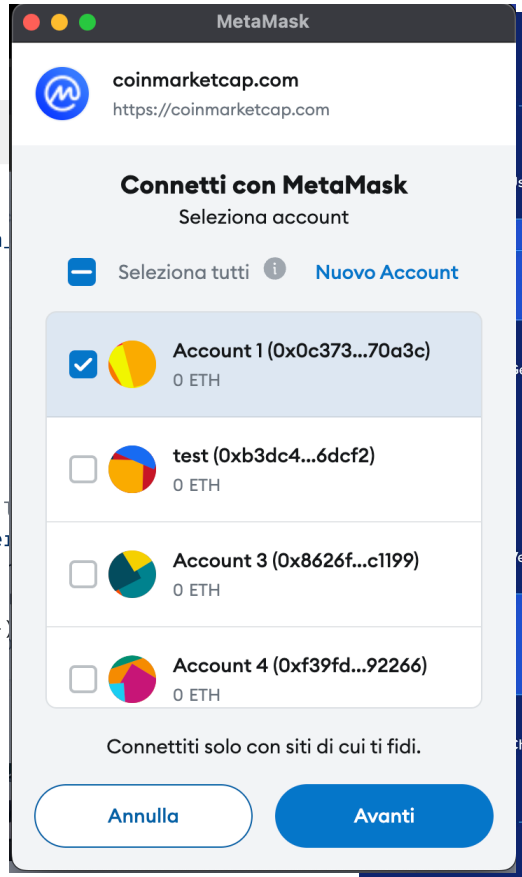
::.. Sign Nonce

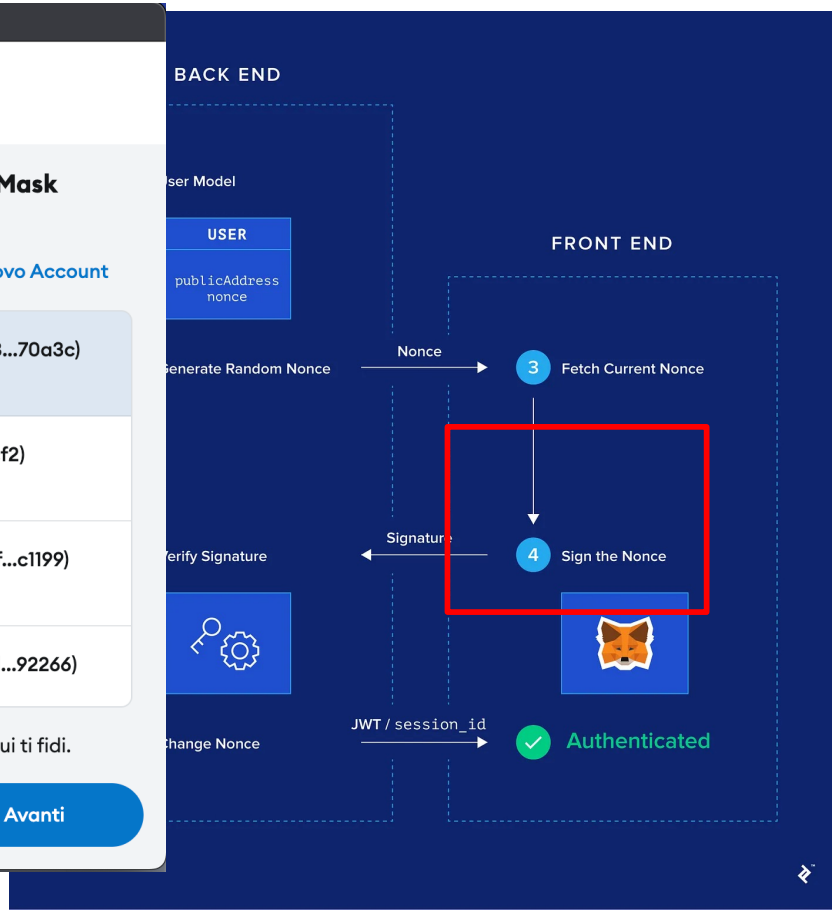
A questo punto si può usare il metodo request per effettuare una richiesta di inidirzzo «eth_requestAccounts» ed una richiesta di «personal_sign», passando come parametro l'indirizzo e il nonce.

```
// Richiede l'indirizzo dell'account dall'utente
const accounts = await ethereum.request({ method: "eth_
const account = accounts[0];

// L'utente firma il nonce
const signature = await ethereum.request({
    method: "personal_sign",
    params: [nonce, account],
});

// Invia il nonce, la firma e l'account al server per l
const response = await fetch("http://localhost:3000/ver
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ account, signature, nonce }),
});
```





::.. Send Signed Nonce

Ora è possibile inviare il nonce firmato al server.

```
// Richiede l'indirizzo dell'account dall'utente
const accounts = await ethereum.request({ method: "eth_requestAccounts" });
const account = accounts[0];

// L'utente firma il nonce
const signature = await ethereum.request({
    method: "personal_sign",
    params: [nonce, account],
});

// Invia il nonce, la firma e l'account al server per la verifica
const response = await fetch("http://localhost:3000/verify", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ account, signature, nonce }),
});
```

::.. Verify Signed Nonce

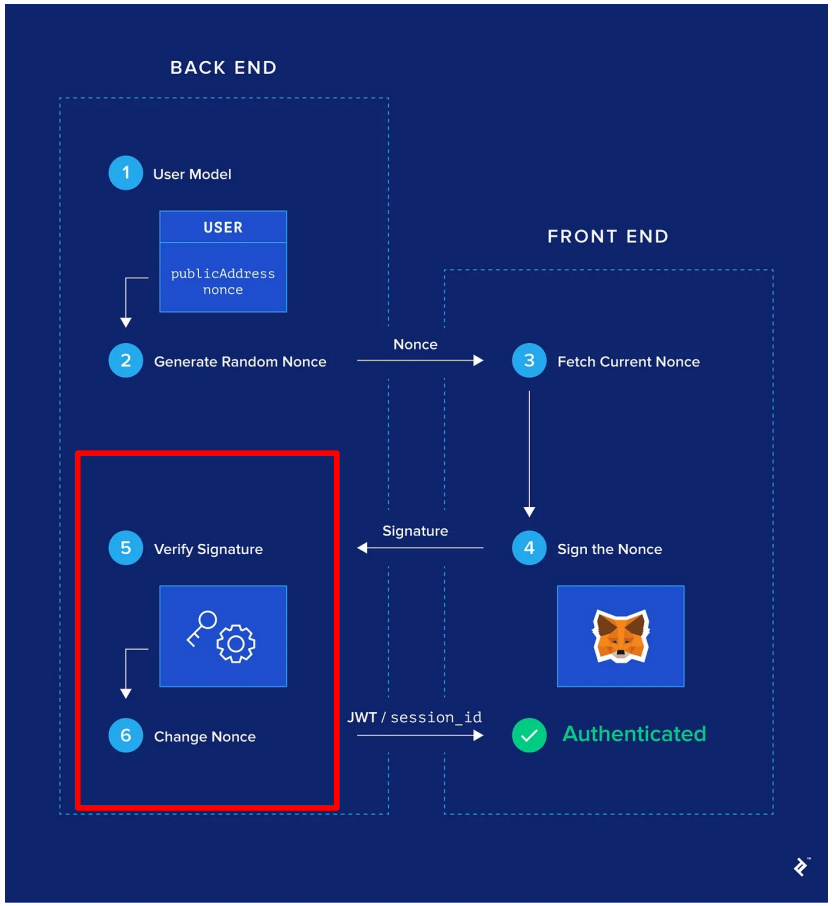
Il server deve implementare un metodo «verify» per gestire la richiesta di autenticazione.

```
app.post("/verify", (req, res) => {
    const { account, signature, nonce } = req.body;

    // Controlla se il nonce è valido
    const index = release.indexOf(nonce);
    if (index > -1) {
        array.splice(index, 1);
    } else {
        res.status(401).json({ success: false, error: "Nonce non valido" });
        return;
    }

    // Recupera l'indirizzo dall'utente che ha firmato il nonce
    const msgBufferHex = bufferToHex(Buffer.from(nonce, "utf8"));
    const recoveredAddress = recoverPersonalSignature({
        data: msgBufferHex,
        sig: signature,
    });

    // Verifica se l'indirizzo è corretto e autorizza la richiesta
    if (recoveredAddress.toLowerCase() === account.toLowerCase()) {
        res.json({ success: true, message: "Accesso autorizzato per l'indirizzo " + account });
    } else {
        res.status(401).json({ success: false, error: "Firma non valida" });
    }
});
```



::.. Access Resource

Se la firma è valida, il server deve rispondere con la risorsa richiesta. Nell'esempio, la risorsa è il messaggio «Accesso autorizzato per l'inidirizzo…».

```
app.post("/verify", (req, res) => {
    const { account, signature, nonce } = req.body;

    // Controlla se il nonce è valido
    const index = release.indexOf(nonce);
    if (index > -1) {
        array.splice(index, 1);
    } else {
        res.status(401).json({ success: false, error: "Nonce non valido" });
        return;
    }

    // Recupera l'indirizzo dall'utente che ha firmato il nonce
    const msgBufferHex = bufferToHex(Buffer.from(nonce, "utf8"));
    const recoveredAddress = recoverPersonalSignature({
        data: msgBufferHex,
        sig: signature,
    });

    // Verifica se l'indirizzo è corretto e autorizza la richiesta
    if (recoveredAddress.toLowerCase() === account.toLowerCase()) {
        res.json({ success: true, message: "Accesso autorizzato per l'indirizzo " + account });
    } else {
        res.status(401).json({ success: false, error: "Firma non valida" });
    }
});
```

::.. Authenticated Session (Statefull)

Tuttavia, affinché questo sistema funzioni, l'utente dovrebbe firmare ogni volta che desidera fare una richiesta autenticata al server.

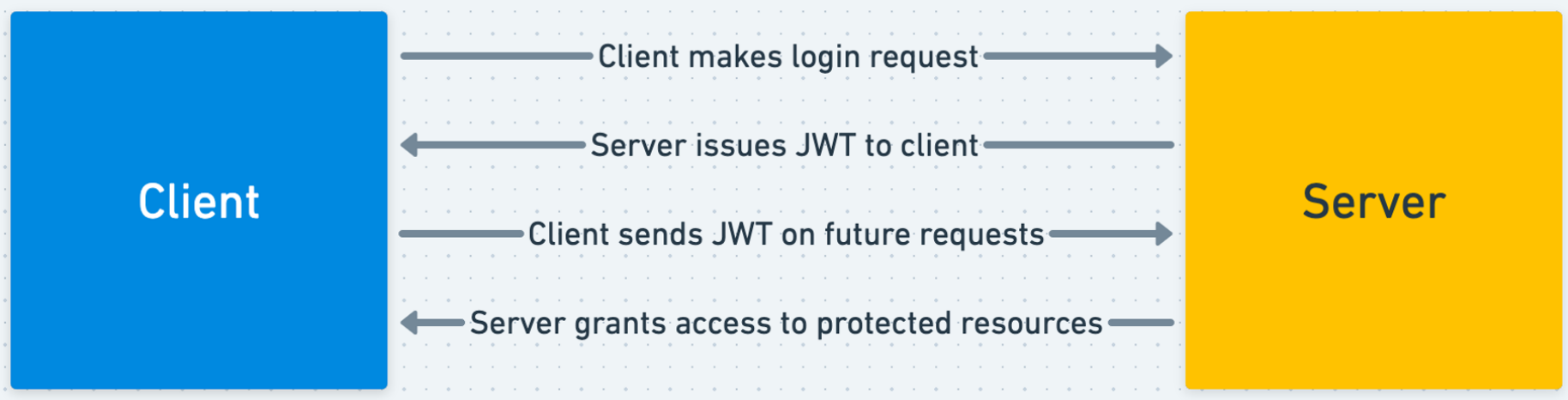
Per risolvere questo problema, l’autenticazione tradizionale utilizza piccoli pacchetti di dati chiamati JSON web token (JWT) per memorizzare le informazioni di sessione dell'utente tra le varie richieste.

Questo permette al server di accedere in modo sicuro all'identità dell'utente e di “ricordare” che l'utente ha già effettuato l'accesso una volta, senza obbligarlo a dimostrare la propria identità per ogni richiesta.

::.. Authenticated Session with JWT

Per ottenere il token JWT è necessario che l'utente dimostri la propria identità (tramite messaggio firmato con il wallet).

Una volta dimostrata l'identità il server rilascia il token che verrà utilizzato per tutte le richieste successive.



::.. Authenticated Session with JWT

E’ possibile modificare l’endpoint di «verify» in modo da rilasciare un JWT anziché le risorsa.

Nella parte alta, la parte relativa alla precedente autenticazione.

Nella parte bassa, l’autenticazione con rilascio di JWT.

```
// Accesso a singola risorsa
if (recoveredAddress.toLowerCase() === account.toLowerCase()) {
    res.json({ success: true, message: "Accesso autorizzato per l'indirizzo " + account });
} else {
    res.status(401).json({ success: false, error: "Firma non valida" });
}

const jwt = require("jsonwebtoken");
const SECRET_KEY = "supersegreto"; // Segreto per firmare il JWT

// Rilascio di JWT
if (recoveredAddress.toLowerCase() === account.toLowerCase()) {
    const token = jwt.sign({ account }, SECRET_KEY, { expiresIn: "1h" });
    res.json({ success: true, token });
} else {
    res.status(401).json({ success: false, error: "Firma non valida" });
}
```

Nell’esempio, il JWT è firmato con una chiave segreta, la cui parte pubblica non è presente in alcun registro, non permettendone la verifica da parte di altri enti, ma permettendone la verifica solo possedendo l’header del JWT.

::.. Released JWT

E’ possibile modificare l’endpoint di «verify» in modo da rilasciare un JWT anziché le risorsa.

La parte alta è relativa alla precedente autenticazione.

Nella parte bassa, l’autenticazione con rilascio di JWT.

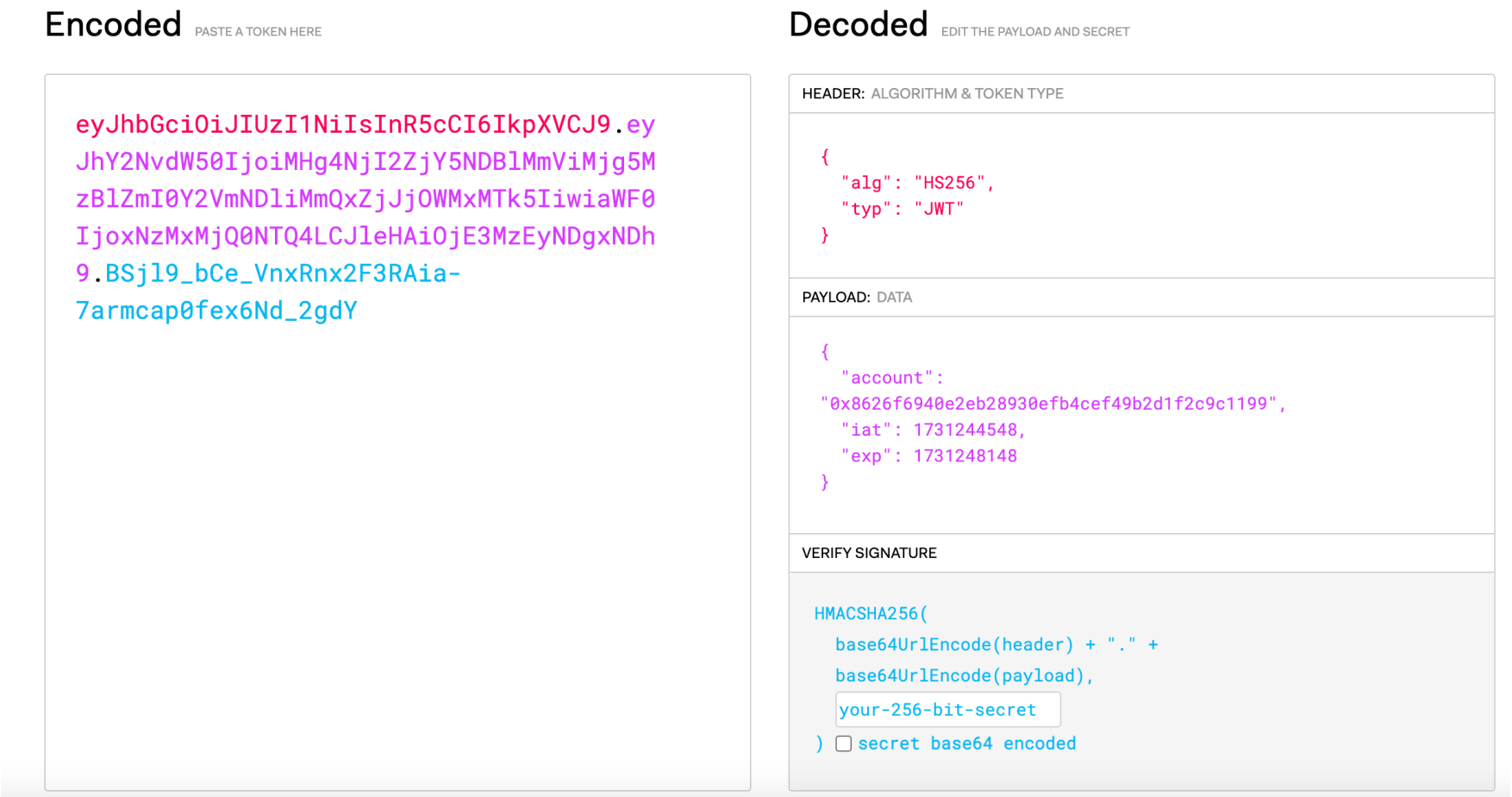
```
// Accesso a singola risorsa
if (recoveredAddress.toLowerCase() === account.toLowerCase()) {
    res.json({ success: true, message: "Accesso autorizzato per l'indirizzo " + account });
} else {
    res.status(401).json({ success: false, error: "Firma non valida" });
}

const jwt = require("jsonwebtoken");
const SECRET_KEY = "supersegreto"; // Segreto per firmare il JWT

// Rilascio di JWT
if (recoveredAddress.toLowerCase() === account.toLowerCase()) {
    const token = jwt.sign({ account }, SECRET_KEY, { expiresIn: "1h" });
    res.json({ success: true, token });
} else {
    res.status(401).json({ success: false, error: "Firma non valida" });
}
```

Nell’esempio, il JWT è firmato con una chiave segreta, la cui parte pubblica non è presente in alcun registro, non permettendone la verifica da parte di altri enti, ma permettendone la verifica solo possedendo l’header del JWT.

::.. Released JWT



Il contenuto del token JWT è strutturato in 3 sezioni: header, payload e verify signature. Nel payload è contenuto l'indirizzo dell'account.

::.. Verify JWT

E’ possibile includere il token JWT rilasciato negli header delle successive richieste, e verificarne la validità nel back-end, prima di dare accesso all’area riservata.

```
// Pagina protetta
app.get("/dashboard", verifyToken, (req, res) => {
    res.render("dashboard", { account: req.account });
});


// Middleware per verificare il token JWT
function verifyToken(req, res, next) {
    const token = req.headers.authorization?.split(" ")[1];
    console.log(req.headers);
    if (!token) return res.redirect("/login");

    jwt.verify(token, SECRET_KEY, (err, decoded) => {
        if (err) return res.redirect("/login");
        req.account = decoded.account;
        next();
    });
}
```