🧱 IRIS-v4 Web3 Upgrade – **Complete End-to-End Implementation Playbook**

> A **single, copy-paste-ready** engineering document that turns the existing iris-v4 repository into a **production-grade AI × Web3** stack.
> Each **section = branch-able GitHub issue** with exact file paths, code snippets, environment variables, and acceptance tests.

🎯 Product North-Star (keep this on the wall)

> “Any end-user can spawn an agent that **reads Ethereum state, signs transactions via Web3Auth, mints memory-NFTs, and accepts USDC payments**—all through the existing Flutter / Streamlit UI.”

📦 0. Repo Add-Ons (git-ready)

```
git checkout -b feat/web3
mkdir -p mcp-servers/web3-mcp
mkdir -p mcp-servers/payments-mcp
mkdir -p smart-contracts
touch smart-contracts/AgentMemoryNFT.sol
touch smart-contracts/PaymentEscrow.sol
```

🔐 1. Web3-MCP (Ethereum Gateway)

1.1 Dockerfile skeleton

```
# mcp-servers/web3-mcp/Dockerfile
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["uvicorn", "server:app", "--host", "0.0.0.0", "--port", "9005"]
```

1.2 requirements.txt

```
fastapi==0.111
uvicorn[standard]
web3==6.*
python-dotenv
httpx
```

1.3 server.py (MCP tool endpoints)

```python
# mcp-servers/web3-mcp/server.py
import os, json
from fastapi import FastAPI
from web3 import Web3
from pydantic import BaseModel

w3 = Web3(Web3.HTTPProvider(os.getenv("ETHEREUM_RPC_URL")))
app = FastAPI()

class Addr(BaseModel):
    address: str

@app.post("/get_eth_balance")
def get_eth_balance(a: Addr):
    bal = w3.eth.get_balance(Web3.to_checksum_address(a.address))
    return {"wei": str(bal), "eth": str(Web3.from_wei(bal, "ether"))}

class TxReq(BaseModel):
    to: str
    value: str        # in ether
    data: str = "0x"
    from_address: str

@app.post("/prepare_transaction")
def prepare_tx(req: TxReq):
    tx = {
        "to": Web3.to_checksum_address(req.to),
        "value": Web3.to_wei(req.value, "ether"),
        "data": req.data,
        "gas": 21000,
        "maxFeePerGas": w3.eth.gas_price,
        "maxPriorityFeePerGas": w3.to_wei("2", "gwei"),
        "nonce": w3.eth.get_transaction_count(req.from_address),
        "chainId": int(os.getenv("CHAIN_ID", 1)),
    }
    return {"unsigned_tx": tx}
```

1.4 Register tool in agent_manager

```python
# agent_api/core/agent_manager.py  (in _create_dynamic_agent_instance)
agent_mcp_config["web3"] = {
    "url": "http://web3-mcp:9005/mcp/" if not local_mode else
"http://localhost:9005/mcp/",
    "transport": "streamable_http"
}
```

💳 2. Payments-MCP (USDC micro-charges)

Repeat docker / requirements pattern as above.
Tool signature:

```
@app.post("/check_balance")
def check_balance(a: Addr):
    usdc = w3.eth.contract(address=os.getenv("USDC_ADDRESS"), abi=ABI)
    bal = usdc.functions.balanceOf(a.address).call()
    return {"usdc_wei": str(bal)}

@app.post("/prepare_payment")
def prepare_payment(req: TxReq):  # value in USDC decimals
    usdc = w3.eth.contract(address=os.getenv("USDC_ADDRESS"), abi=ABI)
    decimals = usdc.functions.decimals().call()
    tx = usdc.functions.transfer(
        Web3.to_checksum_address(req.to),
        int(float(req.value) * 10 ** decimals)
    ).build_transaction({
        "from": req.from_address,
        "nonce": w3.eth.get_transaction_count(req.from_address),
        "gas": 100_000,
        "maxFeePerGas": w3.eth.gas_price,
        "maxPriorityFeePerGas": w3.to_wei("2", "gwei"),
    })
    return {"unsigned_tx": tx}
```

🖼 3. AgentMemoryNFT Solidity

```
// smart-contracts/AgentMemoryNFT.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract AgentMemoryNFT is ERC721URIStorage, Ownable {
    uint256 private _tokenIdCounter;

    constructor() ERC721("AgentMemory", "AM") {}

    function mint(address to, string calldata uri) external onlyOwner returns
(uint256) {
        uint256 tokenId = _tokenIdCounter++;
        _mint(to, tokenId);
        _setTokenURI(tokenId, uri);
        return tokenId;
    }
}
```

Deploy script (Foundry):

```
forge create --rpc-url $RPC --private-key $PK
src/AgentMemoryNFT.sol:AgentMemoryNFT
```

🔑 4. Web3Auth Flutter Integration

4.1 pubspec.yaml

```
dependencies:
  web3auth_flutter: ^2.0.0
  web3dart: ^2.6.1
```

4.2 Minimal service

```
// frontend/cyrene_ui/lib/services/web3_service.dart
import 'package:web3auth_flutter/web3auth_flutter.dart';
import 'package:web3dart/web3dart.dart';

class Web3Service {
  static Future<Web3Client> getClient() async =>
      Web3Client(dotenv.env['RPC_URL']!, Client());

  static Future<Credentials> getCredentials() async {
    final res = await Web3AuthFlutter.login(
      LoginParams(loginProvider: Provider.google),
    );
    final privateKey = res.privKey;
    return EthPrivateKey.fromHex(privateKey);
  }

  static Future<String> signAndSend(Map<String, dynamic> unsigned) async {
    final client = await getClient();
    final cred = await getCredentials();
    final signed = await client.signTransaction(cred, Transaction(
      to: EthereumAddress.fromHex(unsigned['to']),
      value: EtherAmount.fromBigInt(
          EtherUnit.wei, BigInt.parse(unsigned['value'])),
      gasPrice: EtherAmount.fromBigInt(
          EtherUnit.wei, BigInt.parse(unsigned['maxFeePerGas'])),
      maxGas: unsigned['gas'],
      nonce: unsigned['nonce'],
    ));
    return await client.sendRawTransaction(signed);
  }
}
```

🔄 5. End-to-End User Flow

1. User connects wallet in Flutter → `Web3Service.getCredentials()`
2. Agent prepares tx via web3-mcp → returns `unsigned_tx`.
3. Flutter receives tx via WebSocket → `signAndSend()` → tx hash broadcast.
4. On success, agent calls `mint_memory_nft(metadata, user_wallet)` → NFT appears in gallery.

🧪 6. Environment Matrix

Var	Service	Example
`ETHEREUM_RPC_URL`	web3-mcp	`https://eth-mainnet.g.alchemy.com/v2/abc`
`CHAIN_ID`	web3-mcp	`1`
`USDC_ADDRESS`	payments-mcp	`0xA0b86a33E6441b5A3bd7e3bF77bB6b34dE0A36B6`
`NFT_CONTRACT_ADDRESS`	web3-mcp	`0x123...`
`NFT_STORAGE_API_KEY`	web3-mcp	`eyJ...`

✅ 7. Acceptance Tests

Test	Command	Expected
balance	`curl -X POST localhost:9005/get_eth_balance -d '{"address":"0xd8dA…"}'`	`{eth: "1.23"}`
prepare tx	`curl -X POST localhost:9005/prepare_transaction -d '{"to":"0xAbC…","value":"0.01","from_address":"0xUser"}'`	`unsigned_tx`
mint NFT	call `mint_memory_nft` → sign tx → verify on OpenSea	
payment	pay 1 USDC → agent unlocks premium feature	

🏁 8. Ship & Market

- Tag: `git tag v4-web3-beta1`
- Docs: `docs/web3.md` with GIF of wallet flow
- Tweet: "iris-v4 agents now read Ethereum, mint NFTs, and accept USDC 🚀"

You now have a **zero-to-one playbook** to convert iris-v4 into a **live AI × Web3 product**.
Just open the checklist, pull the branch, and **start shipping**.