# Experiment No-6

**Title:** Building a Decentralized Application (dApp) with Web3.js or Ethers.js

**Aim:** To design and build a simple decentralized application (dApp) that connects to the Ethereum blockchain using Web3.js or Ethers.js.

## Theory:

- **dApp (Decentralized Application):**

A software application that runs on a blockchain instead of a centralized server.

- **Smart Contract:**

A program written in Solidity that executes automatically when certain conditions are met.

- **Web3.js and Ethers.js:**

JavaScript libraries used to interact with Ethereum blockchain from a website.

  - **Web3.js** - older, widely used library.
  - **Ethers.js** - lightweight, modern, and easier to use.

- **MetaMask:**

A browser extension wallet that allows users to connect websites to Ethereum blockchain.

- **Working of a dApp:**

  1. User opens a website (frontend).
  2. Website connects to blockchain using Web3.js/Ethers.js.
  3. User interacts with smart contract (e.g., send tokens, check balance).

## Key Characteristics of dApps

1. **Decentralized-** Runs on blockchain, not on a single server.
2. **Transparent -** Anyone can verify the smart contract code.
3. **Trustless -** Works without middlemen.
4. **Secure -**Transactions are cryptographically protected.
5. **Token-based -** Many dApps use ERC-20/ERC-721 tokens.

## Steps of Execution:

1. Write and deploy a **smart contract** on Ethereum Testnet.
2. Build a **frontend (HTML/JS)** for user interaction.
3. Use **Web3.js or Ethers.js** to connect the frontend to Ethereum.
4. Connect the dApp to **MetaMask**.
5. Interact with the deployed contract (read/write data).

## Stepwise Procedure :

### 1. Setup Environment

- Install **Node.js**.
- Install **MetaMask** in browser.
- Create a wallet and connect to **Testnet**.

### 2. Deploy Smart Contract

- Open **Remix IDE**.
- Write a simple contract (e.g., store & retrieve message).
- Deploy it on **Sepolia/Goerli Testnet**.
- Copy the **contract address** and **ABI**.

### 3. Create dApp Frontend

- Make an `index.html` file with a simple UI.
- Add **Web3.js or Ethers.js** library.

### 4. Connect to Blockchain

- Write JavaScript code to connect MetaMask with Web3.js/Ethers.js.
- Load the contract using its ABI and address.

### 5. Interact with Smart Contract

- Create functions in JS to read and write data from the contract.
- Test interaction using MetaMask.

## Program Smart Contract (Solidity – HelloWorld.sol):

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract HelloWorld {
   string public message;

   constructor(string memory _message) {
      message = _message;
   }

   function setMessage(string memory _message) public {
      message = _message;
   }

   function getMessage() public view returns (string memory) {
      return message;
   }
}
```

## Frontend (index.html + Ethers.js Example):

```
<!DOCTYPE html>
<html>
<head>
  <title>My dApp</title>
</head>
<body>
  <h2>Decentralized App Example</h2>
  <p id="currentMessage">Loading...</p>
  <input type="text" id="newMessage" placeholder="Enter new message">
  <button onclick="setMessage()">Update Message</button>

  <script src="https://cdn.jsdelivr.net/npm/ethers/dist/ethers.min.js"></script>
  <script>
    const contractAddress = "YOUR_CONTRACT_ADDRESS_HERE";
    const contractABI = [ /* ABI from Remix */ ];

    async function loadContract() {
      if (window.ethereum) {
        await ethereum.request({ method: "eth_requestAccounts" });
        const provider = new ethers.BrowserProvider(window.ethereum);
        const signer = await provider.getSigner();
        window.contract = new ethers.Contract(contractAddress, contractABI, signer);
        getMessage();
      } else {
        alert("Please install MetaMask");
      }
    }

    async function getMessage() {
      const msg = await window.contract.getMessage();
      document.getElementById("currentMessage").innerText = "Current Message: " + msg;
    }

    async function setMessage() {
      const newMsg = document.getElementById("newMessage").value;
      const tx = await window.contract.setMessage(newMsg);
      await tx.wait();
      getMessage();
    }

    loadContract();
  </script>
</body>
</html>
```

## Key Points :

- dApps combine **frontend (HTML/JS)** + **smart contract (Solidity)**.
- **Web3.js/Ethers.js** helps frontend talk to blockchain.
- **MetaMask** is required for transactions.
- Testnet is used for safe deployment..

## **Conclusion** :

In this experiment, we built a simple decentralized application using Ethers.js. The dApp connected to the Ethereum testnet, interacted with a deployed smart contract, and updated data on the blockchain. This experiment shows how blockchain technology integrates with web applications.

## **Viva Questions:**

1. What is a dApp?
2. Difference between Web3.js and Ethers.js?
3. Why do we use MetaMask?
4. What is an ABI in Ethereum?
5. How does a frontend connect to blockchain?
6. What is the role of a smart contract in a dApp?
7. Why do we first deploy on Testnet?