**1. Solidity (Smart Contracts)**

Solidity

pragma solidity ^0.8.0;  
  
// ERC-20 compatible governance token  
contract GovernanceToken is ERC20 {  
 constructor() ERC20("DACToken", "DCT") {  
 // Initial token distribution (consider more sophisticated mechanisms)  
 \_mint(msg.sender, 1000000 \* 10\*\*18); // Example: Mint 1 million tokens to deployer  
 }  
}  
  
// Membership NFT contract (ERC-1155)  
contract MembershipNFT is ERC1155 {  
 // Define membership tiers (Basic, Contributor, Investor)  
 uint256 public constant BASIC = 0;  
 uint256 public constant CONTRIBUTOR = 1;  
 uint256 public constant INVESTOR = 2;  
  
 constructor() ERC1155("") {}  
  
 function mintMembership(address \_to, uint256 \_id, uint256 \_amount) public {  
 \_mint(\_to, \_id, \_amount, "");  
 }  
}  
  
// DAO Constitution (simplified)  
contract DAOConstitution {  
 // ... Define rules, parameters, and amendment procedures ...  
}

**2. Java (Backend - Spring Boot Example)**

Java

@RestController  
@RequestMapping("/api")  
public class DACController {  
  
 @Autowired  
 private Web3j web3j;   
  
 @Autowired  
 private Credentials credentials;   
  
 @Autowired  
 private ContractLoader contractLoader;   
  
 // Example: Endpoint to mint a membership NFT  
 @PostMapping("/mint-membership")  
 public ResponseEntity<String> mintMembership(@RequestBody MintMembershipRequest request) {  
 try {  
 // Load MembershipNFT contract  
 MembershipNFT membershipNFTContract =   
 ContractLoader.load(  
 contractAddress,   
 MembershipNFT.class,   
 web3j,   
 credentials,   
 GasProvider.getDefaultGasProvider()  
 );  
  
 // Call mintMembership function on the contract  
 TransactionReceipt receipt = membershipNFTContract.mintMembership(  
 request.getToAddress(),   
 request.getMembershipId(),   
 request.getAmount()  
 ).send();  
  
 return ResponseEntity.ok("Membership NFT minted successfully");  
 } catch (Exception e) {  
 return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body("Error minting NFT");  
 }  
 }  
  
 // ... Other endpoints for content management, governance, etc. ...  
}

**3. HTML, CSS, JavaScript (Frontend)**

HTML

<!DOCTYPE html>  
<html>  
<head>  
 <title>DAC Portal</title>  
 <link rel="stylesheet" href="style.css">  
</head>  
<body>  
 <div id="app">  
 <h1>Welcome to the DAC</h1>  
 <button id="mintMembershipButton">Mint Membership</button>  
 <div id="membershipInfo"></div>  
 </div>  
  
 <script src="https://cdn.jsdelivr.net/npm/web3@latest/dist/web3.min.js"></script>  
 <script src="script.js"></script>  
</body>  
</html>  
  
<script>  
 const mintMembershipButton = document.getElementById('mintMembershipButton');  
 mintMembershipButton.addEventListener('click', async () => {  
 try {  
 // ... Interact with the backend API to mint membership NFT ...  
 } catch (error) {  
 console.error('Error minting membership:', error);  
 }  
 });  
</script>

**Important Notes:**

* **This is a highly simplified example.**
* **Security:** Implement proper authentication, authorization, and input validation.
* **Error Handling:** Include robust error handling and user feedback mechanisms.
* **User Interface:** Design a user-friendly and intuitive interface.
* **Blockchain Integration:** Integrate with a suitable blockchain network (e.g., Ethereum, Polygon).
* **Testing:** Thoroughly test all components of the system.
* **Continuous Improvement:** Regularly update and improve the platform based on user feedback and evolving requirements.

This code provides a basic starting point. You'll need to expand upon it significantly to build a fully functional and robust DAC platform. Remember to consult with experienced developers and security experts throughout the development process.

**Sources**

1. <https://github.com/dragon111333/nft-market-pototype>