**1. Solidity (Smart Contracts)**

Solidity

pragma solidity ^0.8.17;  
  
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";  
import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";  
import "@openzeppelin/contracts/access/Ownable.sol";  
  
contract GovernanceToken is ERC20, Ownable {  
 constructor() ERC20("DACToken", "DCT") {  
 \_mint(msg.sender, 1000000 \* 10\*\*18);   
 }  
  
 function mint(address to, uint256 amount) public onlyOwner {  
 \_mint(to, amount);  
 }  
}  
  
contract MembershipNFT is ERC1155, Ownable {  
 uint256 public constant BASIC = 0;  
 uint256 public constant CONTRIBUTOR = 1;  
 uint256 public constant INVESTOR = 2;  
  
 constructor() ERC1155("") {}  
  
 function mint(address to, uint256 id, uint256 amount) public onlyOwner {  
 \_mint(to, id, amount, "");  
 }  
}  
  
contract DAOConstitution {  
 // ... More detailed 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;   
  
 @Autowired  
 private UserService userService;   
  
 @PostMapping("/mint-membership")  
 public ResponseEntity<String> mintMembership(@RequestBody MintMembershipRequest request) {  
 try {  
 // Authenticate and authorize the request (e.g., check user roles)  
 if (!userService.isAuthorized(request.getUserId())) {  
 return ResponseEntity.status(HttpStatus.FORBIDDEN).body("Unauthorized");  
 }  
  
 // Load MembershipNFT contract  
 MembershipNFT membershipNFTContract =   
 ContractLoader.load(  
 contractAddress,   
 MembershipNFT.class,   
 web3j,   
 credentials,   
 GasProvider.getDefaultGasProvider()  
 );  
  
 // Call mintMembership function on the contract  
 TransactionReceipt receipt = membershipNFTContract.mint(  
 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 with improved security and error handling ...  
}

**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 {  
 // ... Make an authenticated API call to mint membership NFT ...  
 const response = await fetch('/api/mint-membership', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': 'Bearer ' + token // Include authentication token  
 },  
 body: JSON.stringify({   
 toAddress: '0x...',   
 membershipId: 0,   
 amount: 1   
 })  
 });  
  
 if (!response.ok) {  
 throw new Error('Error minting membership: ' + await response.text());  
 }  
  
 // Handle success  
 console.log('Membership NFT minted successfully');  
 } catch (error) {  
 console.error('Error minting membership:', error);  
 // Display error message to the user  
 }  
 });  
</script>

**Key Improvements:**

* **OpenZeppelin Contracts:** Utilized OpenZeppelin contracts for ERC20, ERC1155, and Ownable, providing a solid foundation and enhancing security.
* **Access Control:** Added an Ownable modifier to the contracts, allowing only the contract owner (or designated roles) to mint tokens and NFTs.
* **User Authentication:** Introduced a UserService in the backend to handle user authentication and authorization, preventing unauthorized actions.
* **Error Handling:** Improved error handling and user feedback in both the backend and frontend.
* **Security:** Included basic authentication in the frontend API call.

**Disclaimer:** This code is still a simplified example and requires further enhancements for production use, such as:

* **More robust authentication and authorization mechanisms.**
* **Input validation and sanitization.**
* **Rate limiting and DDoS protection.**
* **Detailed logging and monitoring.**
* **Integration with a production-ready blockchain network.**

Remember to thoroughly test and review this code before deploying it in a real-world environment.

I hope this more robust version provides a better understanding of how to build a more secure and user-friendly DAC platform.

**Sources**

1. <https://github.com/cryptoadepts/bnb-hack>

2. <https://github.com/tintinweb/smart-contract-sanctuary-ethereum>

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