**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/AccessControl.sol";  
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";  
  
contract GovernanceToken is ERC20, AccessControl, ReentrancyGuard {  
 bytes32 public constant MINTER\_ROLE = keccak256("MINTER\_ROLE");  
  
 constructor() ERC20("DACToken", "DCT") {  
 \_grantRole(DEFAULT\_ADMIN\_ROLE, msg.sender);  
 \_grantRole(MINTER\_ROLE, msg.sender);  
 }  
  
 function mint(address to, uint256 amount) public onlyRole(MINTER\_ROLE) nonReentrant {  
 \_mint(to, amount);  
 }  
}  
  
contract MembershipNFT is ERC1155, AccessControl, ReentrancyGuard {  
 bytes32 public constant MINTER\_ROLE = keccak256("MINTER\_ROLE");  
  
 uint256 public constant BASIC = 0;  
 uint256 public constant CONTRIBUTOR = 1;  
 uint256 public constant INVESTOR = 2;  
  
 constructor() ERC1155("") {  
 \_grantRole(DEFAULT\_ADMIN\_ROLE, msg.sender);  
 \_grantRole(MINTER\_ROLE, msg.sender);  
 }  
  
 function mint(address to, uint256 id, uint256 amount) public onlyRole(MINTER\_ROLE) nonReentrant {  
 \_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)   
 throws Exception { // Explicitly throw exceptions  
  
 // Authentication and Authorization  
 if (!userService.isAuthorized(request.getUserId(), "MINT\_MEMBERSHIP")) {   
 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");   
 }  
  
 // ... 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) {  
 const errorMessage = await response.text();   
 alert(errorMessage); // Display error message to the user  
 throw new Error('Error minting membership: ' + errorMessage);  
 }  
  
 // Handle success  
 console.log('Membership NFT minted successfully');  
 // Update UI to reflect successful minting  
 } catch (error) {  
 console.error('Error minting membership:', error);  
 // Display error message to the user  
 }  
 });  
</script>

**Key Enhancements:**

* **AccessControl:**
  + Utilized OpenZeppelin's AccessControl for fine-grained access control in smart contracts.
  + Defined MINTER\_ROLE and other roles to restrict access to sensitive functions.
* **ReentrancyGuard:**
  + Incorporated ReentrancyGuard to prevent reentrancy attacks and improve contract security.
* **Improved Error Handling:**
  + More explicit exception handling in Java.
  + Better error message handling and display in the frontend.
* **Enhanced Authentication:**
  + Introduced more granular authorization checks in the backend based on user roles and permissions.
* **User Experience:**
  + Improved user feedback by displaying error messages to the user.

**Disclaimer:**

* This code is still a simplified example and requires further enhancements for production use, such as:
  + **Extensive testing and security audits.**
  + **Integration with a production-ready blockchain network.**
  + **Implementation of advanced security measures (e.g., rate limiting, DDoS protection).**
  + **Detailed logging and monitoring.**
  + **Consideration of gas costs and optimization.**

This final version provides a more robust and secure foundation for your DAC platform. Remember to thoroughly review and test the code before deploying it in a real-world environment.

I hope this comprehensive example is helpful!

**Sources**

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

2. <https://blog.csdn.net/hanru723/article/details/125865854>

3. <https://juejin.cn/post/7324224402304057396>

4. <https://forum.moralis.io/t/getting-nfts-from-erc-721-contract-on-goerli/20503>

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