### **Test Task: Solidity Smart Contract Development**

**Objective:** Create a secure, efficient, and well-tested Solidity smart contract named **"DynamicStakingVault"** that implements staking logic with dynamic rewards distribution, withdrawal fees, and emergency safety features.

### **Requirements:**

#### **1. Basic Functionality**

* Users should be able to deposit an ERC20 token into the staking vault.
* Deposits should be tracked individually for each user.
* Users must be able to withdraw their initial deposit plus accumulated rewards.
* Integration with EIP-4626 vault standard.

#### **2. Reward Calculation**

* Rewards should accrue continuously based on staking duration.
* Rewards distribution must follow a dynamic Annual Percentage Yield (APY) structure:  
  + APY starts at a base rate of **10%**.
  + APY increases dynamically by **0.1%** for every additional **1000 tokens** staked globally in the vault, capped at a maximum APY of **20%**.
  + APY recalculates instantly upon each deposit or withdrawal event.

#### **3. Withdrawal Fees and Lock-up Period**

* Implement a minimum lock-up period of **7 days**. Attempting withdrawal within this period triggers a penalty fee of **5%**.
* Withdrawals after the lock-up period incur no penalty.

#### **4. Security & Emergency Features**

* Include an emergency stop mechanism (Pausable) which halts deposits and withdrawals when triggered by an admin.
* Implement role-based access control (RBAC) using OpenZeppelin’s AccessControl:  
  + Roles required: ADMIN\_ROLE and EMERGENCY\_ROLE.
* Provide functions for emergency withdrawal of tokens by the admin to a predefined secure wallet.

#### **5. Gas Optimization & Best Practices**

* Contract should optimize for minimal gas consumption.
* Use Solidity compiler version ^0.8.x.
* Leverage OpenZeppelin libraries for secure token handling and role management.

### **Deliverables:**

1. **Solidity Smart Contract**
   * Complete source code, clearly commented and documented.
2. **Tests (Hardhat or Foundry)**
   * Comprehensive test cases demonstrating contract functionalities:  
     + Depositing and withdrawing.
     + Dynamic APY calculation under various staking volumes.
     + Enforcement of withdrawal penalties.
     + Emergency pause functionality and access control.
3. **README**
   * Brief documentation explaining:  
     + Your approach to dynamic reward calculation.
     + Security considerations taken into account.
     + Steps to deploy, test, and interact with the smart contract.

### **Evaluation Criteria:**

* Solidity code structure, readability, and maintainability.
* Correct and efficient implementation of dynamic reward logic.
* Robustness of security features and compliance with best practices.
* Quality and completeness of testing suite.

### **Bonus Points:**

* Implementation of an efficient off-chain compatible rewards calculation to reduce gas usage.
* Detailed gas efficiency report comparing different optimizations used.

Good luck!