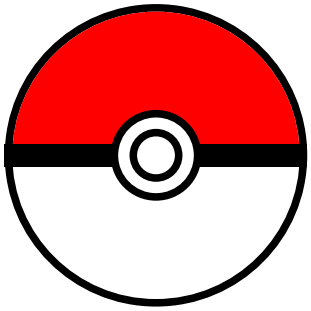
DeFi Coursework 1:
Decentralized Pokémon Card Trading Platform

Arthur Gervais



1 Overview

This coursework assignment challenges students to develop a decentralized application (dApp) for trading Pokémon cards. The project integrates blockchain technology with NFT concepts, requiring implementation of smart contracts, frontend development, and Web3 technologies integration. Students will work in pairs to deliver a complete, functional trading platform deployed on a local testnet.

2 Learning Objectives

The primary objectives of this coursework are to develop practical skills in implementing and deploying smart contracts using Solidity, creating Web3-integrated frontend applications, applying security best practices in DeFi development, understanding NFT standards and token economics, and gaining hands-on experience with development tools such as Hardhat.

3 Project Components and Grading Distribution

3.1 Smart Contracts (40%)

The core of the project consists of smart contracts. First, students must implement an NFT contract for Pokémon cards following the ERC721 standard. This contract should include comprehensive metadata for Pokémon characteristics and implement secure minting functionality with appropriate access controls. The second required contract is a trading contract that enables card listing, supports both fixed-price sales and auctions, and implements secure withdrawal patterns. Students must demonstrate understanding of smart contract optimization and security best practices in their implementation. Students are free to design more than two contracts, but please keep simplicity and clarity in mind.

3.2 Frontend Development (30%)

The frontend component requires development of an intuitive user interface that includes wallet connection functionality, a comprehensive card marketplace, and trading interfaces. The implementation must showcase proper Web3 integration through event listeners for contract updates, transaction handling, and wallet integration.

3.3 Security Considerations (20%)

Security forms a crucial component of the evaluation. Students must address several key security concerns in their implementation. The solution should implement protection against reentrancy attacks. Access control must be properly implemented using function modifiers and role-based access where necessary.

If front-running could be a concern, then students should design and add afront-running prevention (e.g., through the implementation of commit-reveal schemes or similar mechanisms for sensitive operations). Additionally, students should implement protection against integer overflow, proper event emission, and emergency stop functionality.

3.4 Documentation and Presentation (10%)

Students must provide comprehensive documentation including setup instructions and an architecture overview. A **three-minute demo video is required**, demonstrating core functionality and security features while explaining key technical decisions.

4 Technical Requirements

4.1 Development Environment

Students are encouraged to use Hardhat for local blockchain development and testing. The application should be deployed to a local testnet and include a comprehensive test suite. The development environment setup should be clearly documented to ensure reproducibility.

4.2 Code Quality Standards

Code quality will be evaluated based on several criteria: minimization of code redundancy, proper implementation of design patterns, code readability and organization, quality of comments and documentation, and test coverage. Students should follow the official Solidity style guide and implement proper error handling throughout their codebase.

5 Deliverables

The final submission package must include the complete source code for smart contracts, frontend application, test suite, and deployment scripts. Documentation should comprise a detailed README.md with setup instructions, technical documentation. The required three-minute demonstration video should be uploaded along the code and documentation.

6 Group Work Specifications

Students will work in pairs, with clear documentation required for individual contributions. Both team members must demonstrate comprehensive understanding of all project components during evaluation. The distribution of work should be equitable and clearly documented in the project submissions.

7 Optional Enhancements

Students may implement additional features for extra credit, such as Dutch auctions, IPFS integration for metadata storage, advanced trading features including batch trades and swaps, mobile-responsive design, or advanced search and filtering capabilities. These enhancements should not compromise the core functionality or security of the application.

8 Use of GenAI

Students are encouraged to use Cursor or similar applications to enhance their development. These tools are especially excellent at UI work. However, students have to understand their code base deeply. Points will be substracted if it appears that the student do not understand their own code base. Please also do

indicate which AI tools you have used, if you've used them — which is totally fine and legitimate. Just a copy and paste without understanding the content is not appropriate.

9 Technical Support and Resources

Students are encouraged to consult the following resources:

- Hardhat Documentation (`https://hardhat.org/getting-started/`)
- OpenZeppelin Contracts (`https://docs.openzeppelin.com/contracts/`)
- Ethereum Security Best Practices (`https://consensys.github.io/smart-contract-best-practices/`)
- ERC721 Standard Documentation (`https://eips.ethereum.org/EIPS/eip-721`)
- For the demo recording, you can use any software. On MacOS, for example, the default Quicktime software works well.