# **Hub project description sheet**

*Olympe*

## **Project leader(s)**

*LAMBERT Julian – TEK4 – Main Developper*

## **Summary of the project**

## Olympe is a fully decentralized blockchain application (dApp) that allows users to submit and vote on proposals transparently, securely, and immutably. The system leverages Ethereum smart contracts to ensure the voting process is tamper-proof and decentralized. Users connect to the application using their Web3 wallets (e.g., MetaMask) through decentralized authentication by signing cryptographic messages. The goal is to provide a robust and simple decentralized voting mechanism, including proposal management, real-time result tracking, and protection against multiple votes. The project will feature a lightweight web interface, designed following eco-design principles to minimize its environmental impact.

## **Functional purpose**

As a user, I want to:

* Connect with my Web3 wallet (e.g., MetaMask) to interact with the application securely.
* Submit proposals for voting.
* Vote on submitted proposals.
* View the results of the voting process in real time.

As an administrator, I want to:

* Manage voting session (open/close proposals).
* Ensure that only authorized users can vote and that they cannot vote more than once per session.
* Have additional rights to manage administrative operations on the blockchain.

## **Technical / technological runtime**

Material (if needed):

No specific material is required; the project can run locally using Ganache for development testing or be deployed on a testnet like Goerli.

Programming Languages:

Solidity (for writing smart contracts)

JavaScript/TypeScript (for the frontend with Web3.js or Ethers.js for blockchain interaction)

HTML/CSS/JavaScript (for a lightweight web interface)

Resources (API, course material, documentation links):

<https://docs.soliditylang.org/en/v0.8.28/>

<https://web3js.readthedocs.io/en/v1.10.0/>

<https://docs.metamask.io/>

https://solar.lowtechmagazine.com/

## **Deliverable Organization and temporality**

* Functionality 1: Basic Voting System with Authentication (4 days)
  + Voting Smart Contract: The contract will allow the creation of a list of proposals and enable voting on them.
  + Authentication via MetaMask: Users authenticate by signing a cryptographic message using their Ethereum wallets (e.g., MetaMask), proving ownership of their address.
  + Frontend Interaction with Web3.js: The frontend will interact directly with the smart contract using Web3.js or Ethers.js to allow users to submit and view votes.
  + Local Testing: Use Ganache to simulate a local blockchain and test the contract.
* Functionality 2: Proposal Submission & Vote Counting (2 days)
  + Proposal Submission: Users submit new proposals directly to the smart contract using the frontend.
  + Real-Time Vote Counting: Votes are tallied live on the blockchain, and users can query the current results at any time using Web3.js
* Functionality 3: Security & Role Management (3 days)
  + Preventing Multiple Votes: Use a mapping (address => bool) to mark whether an address has already voted on a given proposal.
  + Role Management (Admin vs. User): Use the OpenZeppelin AccessControl library to implement role management directly within the smart contract.
* Functionality 4: Security & Role Management (3 days)

Smart Contract Optimization:

* + Gas Optimization: Optimize the smart contract to reduce execution costs. Focus on minimizing storage writes and using efficient data structures.
  + Optimization Techniques: Avoid unnecessary storage writes. Use smaller data types like uint8 where appropriate. Set function visibility appropriately ( internal vs. public ).

Deployment on Goerli:

* + Deploy the contract on a testnet like Goerli to simulate real conditions. Collect feedback on performance and resolve any potential issues.
* Functionality 5: Lightweight Web Interface (3 days)
  + Interface Design: The web interface will follow eco-design principles, prioritizing minimal resource usage, fast load times, and accessibility.
  + Technologies: Use HTML, CSS, and JavaScript to build a lightweight, responsive web interface
  + User Experience: Ensure the interface provides smooth interactions for proposal submission, voting, and result viewing.

## **Project Recap**

**Total estimated time in days:**

**15 days in total**

**TOTAL XP: *do not fill***

**Stages of the project**

*Break down the different stages of your project into releases. Release 1.0 corresponds to 25% of your project, release 2.0 to 50% of your project, release 3.0 to 75% of your project and release 4.0 to 100% of your project expectations.*

***You need to specify what each release will contain.***

* Release 1.0:
* We will have this stage or features ready :
  + Basic smart contract for managing votes.
  + Frontend with MetaMask authentication
  + Local testing using Ganache.
* Release 2.0:
* We will have this stage or features ready :
  + Proposal submission functionality.
  + Fully functional voting process.
  + Real-time vote counting implmented
* Release 3.0:

Security checks to prevent multiple voting.

Basic optimization for gas efficiency.

System ready for testing on a testnet.

* Release 4.0:

Full testing on a public testnet (Goerli).

Final bug fixes and optimizations.

Project ready for delivery.

## **Delivery**

Screenshots (mandatory): Screenshots of MetaMask interactions (authentication, proposal submission, voting, results).

Light description (mandatory): Explanation of project steps, developed features, and workflow.

Video (optional): Video demonstrating the voting process and result checking via the interface.

Playable project folder: Instructions for running the project locally (with Ganache) or on a testnet like Goerli.