# A green and white logo Description automatically generated

# **Blockchain Technologies and Applications**

**Block 8**

**Due Date: Week 10, Friday 1pm.**

**Version: 1.1, 03/04/2024**

## Objectives

* Learn the mechanics of a blockchain distributed application.
* Gain a foundation in Solidity, a smart contract programming language.
* Through practice, gain a appreciation of the applications of blockchain beyond cryptocurrency.

## Description

Create a Web3 Distributed Application (DApp) that implements simple ticketing system. You should use the Ethereum Sepolia Testnet as you blockchain for Solidity smart contract deployments, and HTML, CSS and Javascript for your front end. You are free to use any development environments that you are comfortable with, however examples in this module will be demonstrated using Visual Studio Code and the Online Remix Solidity compiler and deployment tool.

## Requirements

* Front End:
  + Page allowing a user to create a wallet.
    - Should provide the ability to download the created wallet.
    - Should display wallet details once created.
  + Page allowing a user to check their current crypto and ticket token balance.
    - To be used by the following actors:
      * Person attending the event so that they can confirm their purchase.
      * Doorman, so they can confirm a wallet is the holder of the ticket.
      * Venue, so they can check on distribution of tickets
  + Page allowing a user to buy a ticket (token).
  + Page allowing a user to transfer a ticket back to the vendor.
* Blockchain Backend:
  + Smart contract implementing the ERC-20 standard and extended to allow tickets to be purchased using the native cryptocurrency of Sepolia (SETH)
* Project should be submitted as a zipped solution
* The project should be accompanied by a report detailing the following:
  + Code overview.
  + Design description.
  + Links to transactions on Sepolia’s blockchain explorer showing:
    - A successful deployment of your contract
    - A successful execution of your contract to buy a token
    - A successful topping up of separate wallets for:
      * Contract creator
      * Ticket Purchaser
      * Vendor / Doorman
* A video of my reference implementation of the front end can be found here.

## Submission Guidelines

You should submit your entire project as a zipped archive containing:

* All programming artefacts (html, css, js, Solidity Files)
* Details of where your smart contracts are deployed to (contract addresses, holding wallets)
* Instructions on how to run your project. Preferably you should be able to open your project folder in Visual Studio Code and be able to run the project from there using the Live Server plugin
* A project report describing you project

Project should be submitted via Brightspace.

Plagiarism will not be tolerated.

## Marking Scheme

#### **Front End (40%)**

* **Wallet Creation Page (10%)**
  + Ability to create a wallet: **5%**
  + Ability to download the created wallet: **2.5%**
  + Display of wallet details: **2.5%**
* **Balance Check Page (10%)**
  + Functionality for different actors (attendee, doorman, venue): **5%**
  + Accuracy of balance display: **5%**
* **Ticket Purchase Page (10%)**
  + User-friendly interface: **3%**
  + Successful transaction process: **7%**
* **Ticket Transfer Page (10%)**
  + Ease of transferring tickets back to the vendor: **5%**
  + Correct update of token balance post-transfer: **5%**

#### **Blockchain Backend (40%)**

* **Smart Contract (20%)**
  + Implementation of ERC-20 standard: **10%**
  + Extension for ticket purchase with SETH: **10%**
* **Project Submission (10%)**
  + Correct file structure in zipped solution: **5%**
  + Completeness of all required files: **5%**
* **Report (10%)**
  + Code overview clarity: **3.33%**
  + Design description detail: **3.33%**
  + Links to transactions and their success: **3.34%**

#### **Code Quality and Documentation (20%)**

* **Code Readability (5%)**
  + Commenting and documentation: **5%**
* **Code Efficiency (5%)**
  + Optimization of code: **5%**
* **Error Handling (5%)**
  + Proper error messages and handling: **5%**
* **Code Testing (5%)**
  + Comprehensive tests and coverage: **5%**

Total: **100% (50% of Module grade)**