TTM4195 Smart contracts in Solidity

1 Requirements

This assignment requires you to write a smart contract in Solidity, and does not require you to install any additional software. The assignment should help you understand how smart contracts work and how to deploy them in practice. This assignment should be done in groups of 4 (but we are open to accept smaller groups).

In order to facilitate your entrance into the Solidity programming world, you will find on Blackboard a short introduction to Solidity. Please, refer to the Solidity website for the complete documentation of the latest version (v0.8.9).

For this assignment, you will need two essential tools: an Ethereum wallet and Remix.

Nowadays, we have two families of wallets:

- **hardware wallets**, resembling an hard drive, which typically interface with other devices via USB. They offer physical storage of private keys and digital currencies, which will no longer be stockpiled on online servers. In case you will decide to invest around 1500NOK in one of these devices, please remember to triple-check the reliability of the vendor. Some examples of the (allegedly) safest hardware wallets are Ledger Nano X and Trezor Model T;

- **software wallets**, which are encrypted and require a password to access the coins they collect. You have the ability to restore and recover your coins by utilizing the recovery phrase in the event that you forget your password or your computer is damaged or hacked. For Ethereum, the most popular (*and the one we strongly recommend you to use*) is Metamask, a browser extension that guarantees instant access to the Ethereum network (and to the testnet Ropsten, which you will be using). Another valuable option is Eidoo, from a non-custodian company, which comes in the form of app for iOS, Android and desktop devices.

You will make use of the **Remix IDE**, an open source software which allows you to test, debug and deploy smart contracts written in Solidity or Yul for the Ethereum blockchain. It is available both offline and online, and we strongly recommend you to go for the online version. The interface is pretty simple and it even allows you to decide the compiler version; for any queries see the online documentation.

2 Assessment and deadlines

There are two deadlines for this assignment:

1. *Friday 19 November, 23:59:59*, written report and code delivery.
   The group task is to implement the smart contract outlined in Section 4. The individual task is to write a half page report in which each member describes his/her own experience in carrying out the assignment. The write-up should discuss (without being limited to) the obstacles in team-working, implementing and distributing the workload. Each individual document will not be shared with the rest of the group. Both the code and the report must be uploaded using the submission page on Blackboard.

2. *Tuesday 23 (16:00-18:00) and Friday 26 (12:00-14:00) November*, online presentation. The whole team will have to explain how the smart contract works and the implementation strategies (splitting of the workload, obstacles and difficulties, ...). 10 minutes will be allocated for each presentation, followed by up to 5 minutes for questions on your implementation rationale (you will be asked to explain in details one of the functions/objects in your implementation).

Time-slots allocation will be performed on a first-come-first-served basis until Monday 15 November, 23:59:59. Send an email to Mattia (mattia.veroni@ntnu.no) to reserve a time slot for your group. The schedule of the presentations is attached to the assignment text on Blackboard, and it is constantly updated.

3 Grading

There are 15 marks available for this assignment:

1. **9 points for the code**, deadline on *Friday 19 November, 23:59:59*.
   As reference for the structure of your smart contract, see Section 4. Each step is worth either 1 or 2 points, as specified in brackets, and there is a total of 11 available points. In this way, two extra points can be awarded, in order to make up for possibly lost points (if you missed some from other steps or from the presentation). Your code will be evaluated after the submission deadline.

2. **6 points for the presentation**, *23 and 26 November, during the usual class hours*.
   The presentations will be open to the whole class, and all group members are required to attend. You are encouraged to use slides, but feel free to decide how to present: for instance you can give a live demonstration, or show a recorded run of the smart contract. You will be evaluated on the clarity of presentation and the understanding of the related concepts. Here are the links to the the Zoom meeting on Tuesday 23 November and to the Zoom meeting on Friday 26 November.

4 The smart contract: TicketBookingSystem

The goal is to create a smart contract (or multiple ones, if you think it is convenient) to manage ticket sales for shows in theatre or cinemas, and to maintain a public collection of shows that each user has attended. Let A, B, C, D be the four members in your group. A plays the role of the sales manager at a theatre in Boydway, the most famous theatre distric in Colin Island. B is a customer who buys a ticket for one of the shows, which B attends and gets a unique poster as a proof of attendance, that he/she can proudly showcase. C buys a ticket for a different show, but later on realizes that he/she cannot attend, and sells the ticket to D.

In order to realize this hypothetical scenario, we ask you to create a smart contract **TicketBookingSystem** and two tokens **TICKET** and **POSTER**. The SC is deployed by A to sell tickets for two different shows:

1. (2 pt.) For each show, initialize the smart contract with the title of the show, the available seats and any other relevant information. Each seat is an object that contains at least
   - title and date of the show,
   - the price;
   - the seat number and row;
   - a link to the seat view (to realize a service offered, for example, by seat-plan.com). It does not need to be working, but set up a field for it;
2. (2 pt.) implement a function **buy** that B and C individually call to get a ticket each, which corresponds to a specific show, date and seat. The function generates and transfers a unique ticket upon purchase, as an instance of the **TICKET** token;
3. (1 pt.) implement a function **verify** that allows anyone with the token ID to check the validity of the ticket and the address it is supposed to be used by;
4. (2 pt.) implement a function **refund** to refund tickets if a show gets cancelled;
5. (2 pt.) implement a function **validate** to validate a ticket; it can be called only in a specific time frame, corresponding to a suitable amount of time before the beginning of the show. Upon validation, the ticket is destroyed, and a function **releasePoster** releases unique proof of purchase (you may consider merging these two functions together). This new item must be a unique instance of a **POSTER token**;
6. (2 pt.) implement a function **tradeTicket** that allows C and D to safely trade (i.e. exchange for another or sell for ether) a ticket directly between each other.