Final Report Template

Galeri Marco
marco.galeri@studio.unibo.it

Patrignani Luca
luca.patrignani3@studio.unibo.it

February 13, 2025

Abstract

Up to ∼2000 characters briefly describing the project (i.e. its goals and results).

1 Goal/s of the project

The project's goal is to implement a decentralized poker game without a trusted third party. In particular our implementation will guarantee that given k players, an adversary cannot rig the game even controlling $k - 1$ players.

Detailed description of the project goal/s. Use case diagrams, examples, or Q/A simulations are welcome.

1.1 Usage scenarios

Informal description of the ways users are expected to interact with your project. It should describe *how* and *why* a user use / interact with the system.

1.2 Definition of done

The delivered artifacts are

- A golang library which implements decentralized deck and some operation on the deck such as shuffling, dealing face down cards and showing a card. The implementation follows the paper by Wei and Wang[4].
- A golang library which implements the actual game of poker
- An CLI interface for playing the game of poker.

The functionalities are mainly tested automatically.

2 Background and link to the theory

(To be written at the end)

- Relevant architectural styles (the ones mentioned in Section 3.2.2)
- Relevant interaction patterns (the ones mentioned in Section 3.2.2)
- Relevant software frameworks (the ones mentioned in Sections 3.2.2 and 4)

3 Requirements Analysis

3.1 Functional requirements

The player is able to play a game of poker:

1. The player will be shown two cards, which are not seen by any others players
2. When it is his turn the player can call, raise or fold
3. When it is the big blind's turn or the raising player's turn checks, three cards will be dealt face up
4. These operations are repeated three more times: the first two times one card is dealt face up, the last time every player shows their own cards. Then the winner is determined
5. if only one player is still in the game the game ends

3.2 Non functional requirements

3.2.1 Deck's requirements

Each of the following requirements have to be met even in the condition that given k players a coalition of $k-1$ players are trying to manipulate the game without that the one out is not informed of the manipulation.

- When shuffling the deck no player can control the final order of the cards
- When the deck is shuffled no player knows the order of the cards
- All cards are present in the deck exactly once
- When a card is dealt to a player faced down, no other players knows which card it is
- When a player shows its card, the other players can verify that he is not lying

3.2.2 Poker game's requirements

- The game uses a deck which meets the requirements previously described
- When player makes a move (call, check or fold), he cannot later repudiate its choice.

3.3 Top-down analysis

3.3.1 Deck implementation

Due to the decentralized nature of the project every communication has to be considered a *broadcast*, that is a player can only communicate with other players by sending the same information to every players.

- What architectural style / interaction pattern / software framework are better suited to develop the project
- ... and why w.r.t. requirements

4 Design

The design process has been a mix between top-down and bottom-up approaches. The solution was designed to have three main components:

- The deck
- The DLT
- The game of poker.

This is where the logical / abstract contribution of the project is presented. Always remember to report **why** a particular design has been chosen. Keep the discussion *technology-free* as much as possible. Reporting wrong design choices which has been evaluated during the design phase is welcome too.

4.1 Structure (domain entities)

Which entities need to by modelled to reflect the domain? UML class diagram here with domain entities and possibly messages being exchanged

4.2 Interaction

As already previously mentioned, each player can communicate only via two primitives:

- `broadcast`: a player sends some information to every other players

- `all-to-all broadcast`: every players sends some information to every other players

Moreover our two operations will implicitly behave as a barrier synchronization, such that they will stop the thread execution until every nodes have terminated the communication. This choice has been made while studying the already mentioned Wei and Wang paper[4]: their protocols only use these two primitives.

4.2.1 The high performance computing similarity

The aforementioned primitives are inspired from the *Message Passing Interface*[3], an open library standard for distributed memory parallelization widely used in high performance computing. From those who are familiar with MPI our two primitives can be seen respectively as `MPI_BCAST` and `MPI_ALLTOALL`. We also observed that the protocols described in the paper are executed by every nodes. This resembles the computational model *single program, multiple data*[5]. This is the most used model in parallel programming and it states that all nodes run the same program but each node may choose a different execution path depending on its id. We believe that this model can be applied also to decentralized systems such ours.

How should entities interact with each others? UML activity / sequence diagram and protocols definitions.

4.3 Behaviour

How should each entity behave? UML State diagram.

4.4 Architecture

How are software pieces organised into software modules? UML component / package / deployment diagrams, data-flow among components, web API description.

4.5 Corner cases

- Faults detection
- Recover strategies
- Error messages
- Graceful shutdown

5 Salient implementation details

Anything potentially interesting / non-trivial and technologies adopted to match the design. This section is expected to be short in case some documentation

(e.g. Javadoc or Swagger Spec) has been produced for the software artefacts. This this case, the produced documentation should be referenced here.

6 Validation

Choose a criterion for the evaluation of the produced software and **its compliance to the requirements above**. Description of automated (and manual) tests and their rationale. In case of a test-driven development, describe tests here and possibly report the amount of passing tests, the total amount of tests and, possibly, the test coverage.

7 Deployment Instructions

Explain here how to install and launch the produced software artefacts. Assume the software must be installed on a totally virgin environment. So, report **any** configuration step. Gradle and Docker may be useful here to ensure the deployment and launch processes to be easy.

8 Usage Examples

Show how to use the produced software artefacts. Ideally, there should be at least one example for each scenario proposed above.

9 Conclusions

Recap what you did.

9.1 Future Works

Recap what you did *not*

9.2 What did we learned

Recap what did you learned.

Stylistic Notes

Use a uniform style, especially when writing formal stuff: X, X, $\mathbf{X}$, $\mathcal{X}$, $\mathtt{X}$ are all different symbols possibly referring to different entities.

This is a very short paragraph.

This is a longer paragraph (notice the blank line in the code). It composed by several sentences. You're invited to use comments within `.tex` source files to separate sentences composing the same paragraph.

Figure 1: Some floating image

Paragraph should be logically atomic: a subordinate sentence from one paragraph should always refer to another sentence from within the same paragraph.

The first line of a paragraph is usually indented. This is intended: it is the way LaTeX lets the reader know a new paragraph is beginning.

Let LaTeX decide where to put figures (or tables, or listings), label them and reference the labels instead of say things like "in the following image...". Consider for instance the case of fig. 1.

Use the `listing` package for inserting scripts into the LaTeX source. Consider for instance listing 1.

Listing 1: Some Java listing

```
public class HelloWorld {
  public static void main(String[] args) {
    // Prints "Hello, World" to the terminal window.
    System.out.println("Hello, World");
  }
}
```

References

[1] D. Adams. *The Hitchhiker's Guide to the Galaxy*. San Val, 1995.

[2] Giovanni Ciatto, Alfredo Maffi, Stefano Mariani, and Andrea Omicini. Smart contracts are more than objects: Pro-activeness on the blockchain. In Javier Prieto, Ashok Das Kumar, Stefano Ferretti, António Pinto, and Juan Manuel Corchado, editors, *Blockchain and Applications*, volume 1010 of *Advances in Intelligent Systems and Computing*, pages 45–53. Springer, 2020.

[3] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.1*, November 2023.

[4] Tzer-jen Wei and Lih-Chung Wang. A fast mental poker protocol. *IACR Cryptology ePrint Archive*, 2009:439, 05 2012.

[5] Wikipedia contributors. Single program, multiple data — Wikipedia, the free encyclopedia, 2025. [Online; accessed 13-February-2025].