Purpose

The purpose of this Software Design Document is to provide a description of the design of the Thesis Management System. The design is detailed enough to allow software developers to proceed with a good understanding of what is to be built and how it is expected to be built. This Software Design Document also provides necessary information regarding the components of the software which can be listed as the following:

- Database
- User Interface / Web Application
- Student Subsystem
- Coordinator Subsystem
- Supervisor Subsystem
- Reader Subsystem
- Opponent Subsystem

In this design document, no requirements will be listed because none of the requirements is being designed at this stage.

1. General Priorities

Throughout the design process, certain design principles were given a priority. In this section, general design priorities are listed along a brief explanation of why they were prioritized over others.

- Modulatory: the divide and conquer principle helps breaking a big system into smaller subsystems. As a result, separate people can work on different parts of the system. It also makes the system easier to understand and easier to maintain and modify. Modularity also makes concurrent execution of the system possible. This was achieved by dividing the Thesis Management System into:

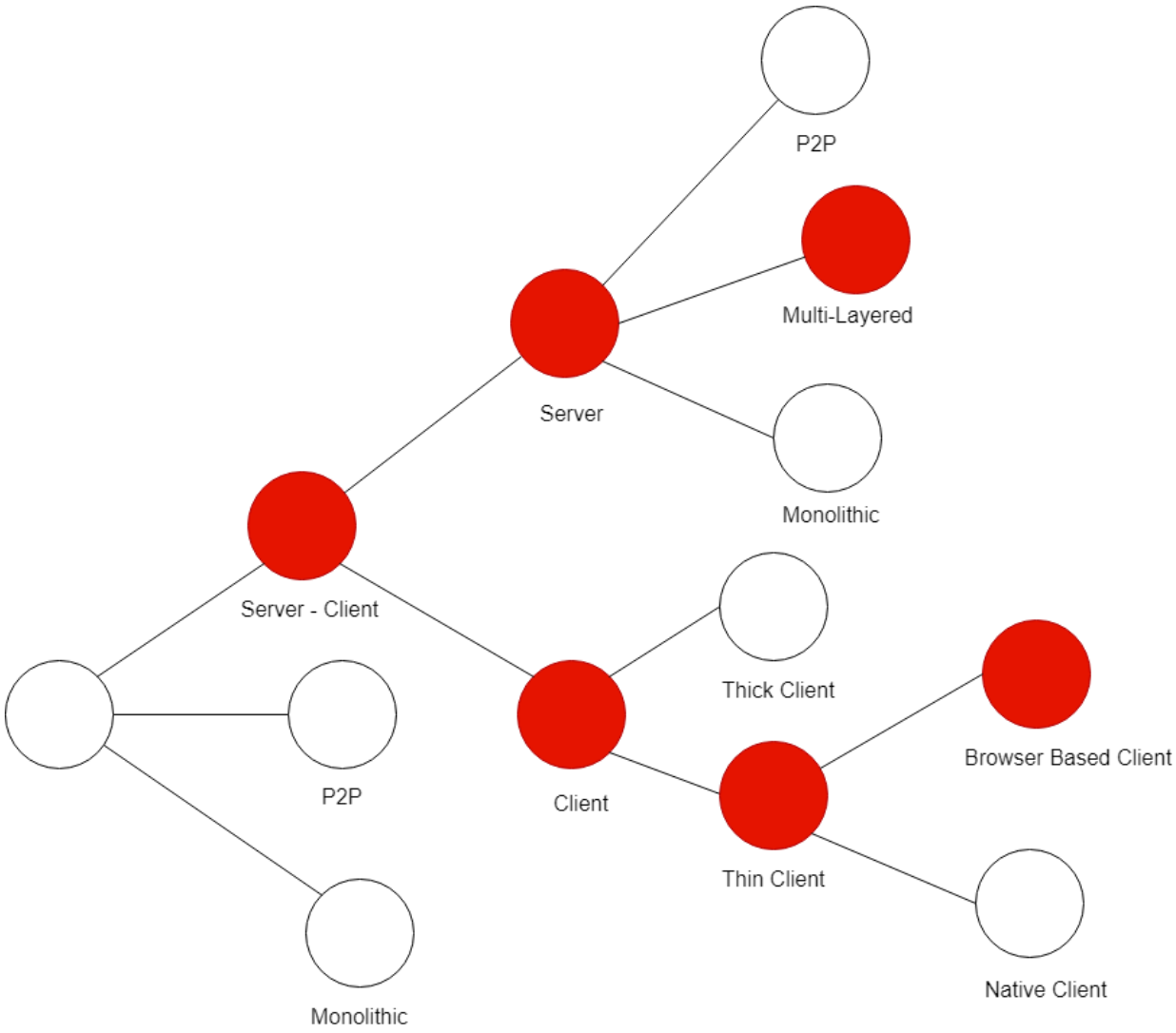
    - Database
    - User Interface / Web Application
    - Student Subsystem
    - Coordinator Subsystem
    - Supervisor Subsystem
    - Reader Subsystem
    - Opponent Subsystem

- Maintainability: As a consequence of modularity, parts of the system can be changed or replaced without replacing or extensively changing the system.
- High level of cohesion: another consequence of modularity that makes the system easier to understand and change. Furthermore, components with high cohesion can be reused again.
- Fault tolerance: the system should continue to operate in case of the failure of one or more of its components. Following a modular design strategy gives the system a good fault tolerance.

- Keeping the high level of abstraction to provide information hiding: by making sure the components of the system only communicate through a given set of instructions (components can access the database but not vice versa).
- Reduce coupling: reducing interdependencies among the different modules by implementing a modular design which makes it easier to modify a component since it does not affect the other components of the system.

2. Design Outline

This section shows a high level description of how the application is designed. It also shows the different alternatives that were considered while the design was taking shape.



- **Server-Client:** a client server architecture was selected as it improves security by ensuring that only authorized clients can access and manipulate data on the servers. Peer 2 Peer was dropped due to its unsuitability for websites. Two or more of the same application act as server and client at the same time towards each other. Monolithic architecture is very fast but not scalable. That means the system has to be ignited every time a client uses it. It also makes it hard to understand the system or even implement changes in a complex application because of the lack of modularity. Thus, it was not considered for this part of the system.

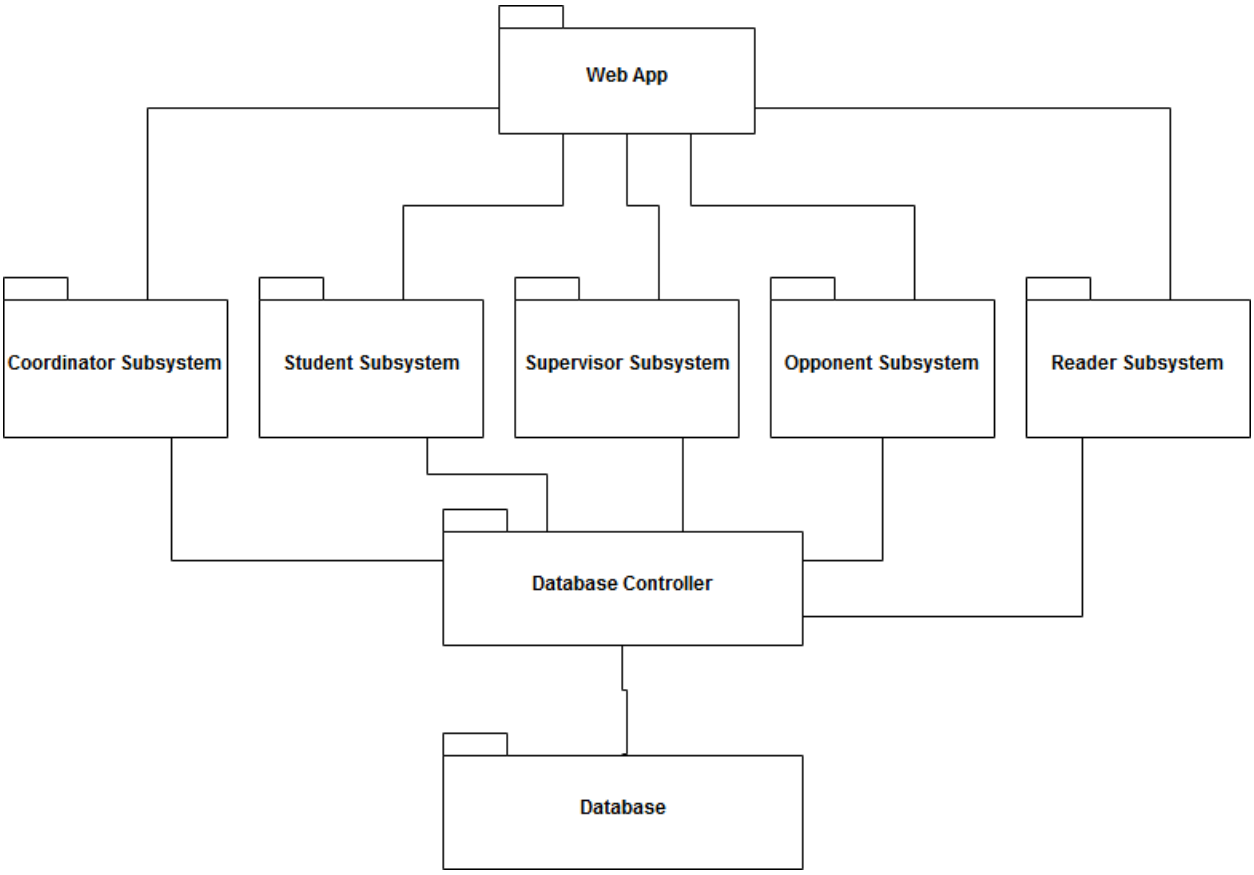
- **Thin client:** The decision to choose this for the design was mainly because as we are making a web application, we need the server to do most of the computational tasks. A thick client means that most of the tasks are executed on a local computer which does not server the purpose and functionality of the Thesis Management System very well.

- **Multilayered Server:** This type of architecture was chosen because it promotes modularity and makes it easier to add new subsystems. Peer 2 Peer architecture was not chosen because it is not suitable for websites, it is difficult to administer and it does not provide good security. Monolithic architecture makes the system hard to understand and implementing changes can be challenging in complex systems. It also creates security and access control problems because in order to restrict access to certain data, a permission based system is used and these systems can be subject to hacking or other kinds of data exploitation. Therefore, monolithic was not considered for the server part.

- **Browser Based Client**: The application is required to be a web application. Therefore, a browser based client is considered to meet that requirement.

3. Software Architecture

The section presents the general architecture of the software. For the specific implementation details, class diagrams, will be delivered during the implementation iteration. Note, however, that this is still work in progress and it is subject to change during the development phase.



The above component diagram is the chosen architecture. When we created the design, we had modularity as a main priority. As such we want the different subsystems to communicate between each other as little as possible. As mentioned previously in the document we chose a combination of server-client and layered patterns. Thus, the software has 4 layers:

1. The Web App layer: this layer represents the client and holds as little application logic as possible.
2. The subsystem layer: the first layer of the server. It is composed of different independent subsystems, and each subsystem takes care of different functions of the system. They are grouped based on the actor they serve. The layers are completely disconnected and independent. So, if, for example, one actor is removed, the corresponding subsystem can be safely removed. Also, it allows adding new subsystems without extensive modifications.
3. The database controller layer: this component is responsible for controlling the data that enters or leaves the database. Thus, it acts as an interface/API for the database.
4. The database layer.