

Sawyer Stanley
0289276
COMP 4911
Bookdex: Design Document
March 4, 2024

Contents

1	Preface	3
2	Overview	3
3	Development Plan	3
4	UML Class Diagrams	4
5	Retrospectives	4
5.1	Week 1	4
6	Decisions	4
6.1	TDS Seperate Implementation	4
6.2	Omitting Timers	5
6.3	Client-side I/O	5

1 Preface

The following document contains justifications and explanations of the decisions made during the development of Bookdex.

Acronyms are often used throughout this document. Common acronyms include:

- **TDS**: Tabular Data Stream
- **GUI**: Graphical User Interface
- **UML**: Unified Modeling Language
- **I/O or IO**: Input/Output

2 Overview

Bookdex is an application from which a user may access a server database of their home library. Note, the user can not read books through Bookdex, only query their existence.

Bookdex makes use of the TDS protocol when transmitting messages between client and server. TDS ensures the secure transmission of SQL queries and the results to those queries. TDS specifications were loosely followed due to time constraints. For more information on which corners were cut, see Section 6.

3 Development Plan

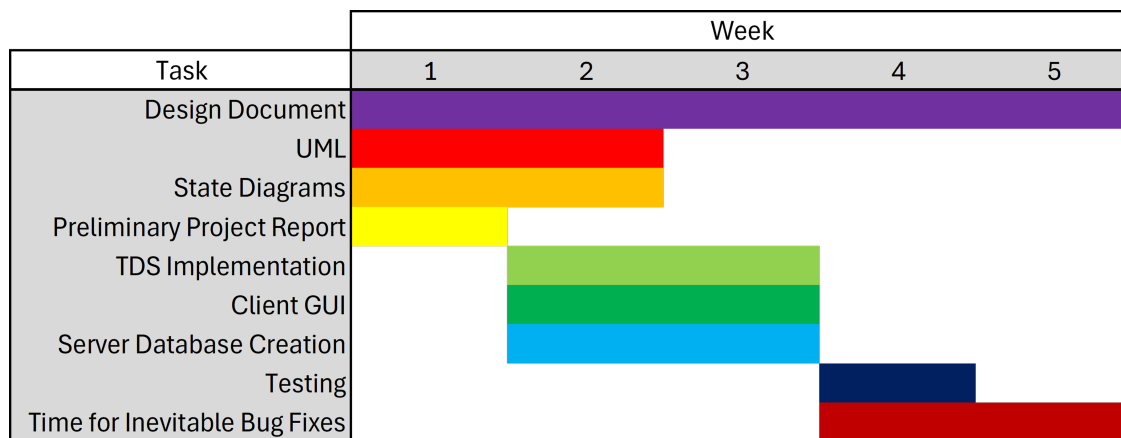


Figure 1: Gantt chart outline for the development plan for Bookdex.

4 UML Class Diagrams

Recognizing that this is difficult to see, .png versions have been uploaded to the [GitHub repository](#) for this project.

The (current) UML class diagram for the TDS protocol is shown in Figure 2.

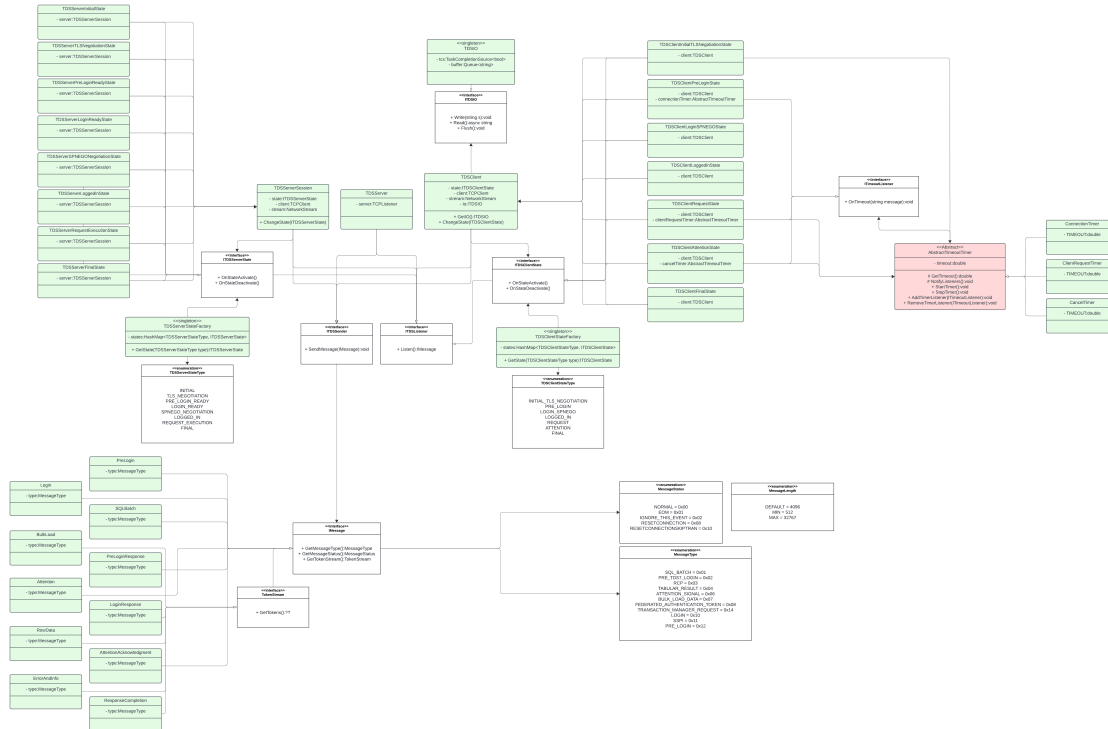


Figure 2: UML diagram outlining the structure of the TDS protocol implementation.

5 Retrospectives

5.1 Week 1

This week began with work on the UML class diagram for the TDS protocol. During its creation, reference **INSERT TDS MANUAL REFERENCE** was consulted regularly. This slowed progress as the TDS protocol is nuanced. For example, there are a handful of different types of tokens to be used when transmitting a token stream, each with their own rules. To add to this, not all messages transmitted are token streams. Decomposing the intricate case-work into meaningful classes has helped with understanding the protocol overall.

6 Decisions

While not every decision is outlined in this section, decisions that were made unexpectedly or creatively are included.

6.1 TDS Seperate Implementation

Ideally, this implementation of the TDS protocol could be bundled and imported into other server database projects. Thus, I have decided to separate its implementation from the main Bookdex implementation. This

is visible in the [project repository](#) as the “bookdex” directory is separate from the “TabularDataStreamProtocol” directory. Hopefully, this approach makes testing TDS easier as it will be modular.

6.2 Omitting Timers

As can be seen in Figure 2, various timers are used by the client to ensure the client is not waiting too long for a response from the server. It’s important to note that TDS does not require these timers. Initial implementation will omit these timers for simplicity’s and time’s sake.

6.3 Client-side I/O

As seen in Section 4, the states in which a TDSClient can exist handle the Listen() operation. Initially, Listen() was intended for receiving packets send from the server to the client. However, when in the TDSCliientLoggedInState, the user should be able to enter a request to be sent to the server before the client listens for a response. The definition of Listen() was then expanded to include “listening” for input from the user.

Anticipating user input to come from various input streams, I planned to create a custom input stream to which all input could be written and from which the program could consistently read. Upon researching halting the program to wait for input on said stream, I began to fear I had gone astray. Instead, I opted to create custom I/O, namely, TDSIO (Tabular Data Stream Protocol Input/Output). Now, when a user writes to the “input stream”, their input is enqueued to a queue of inputs. When these inputs are flushed, a notification of a complete task will be sent to the listening TDSCliientLoggedInState. The queue is then emptied and the TDSCliientLoggedInState can do with the inputs what it will. This way, implementation of a view, whether GUI or console, will be supported (I think).