Sawyer Stanley
0289276
COMP 4911
Bookdex: Design Document
March 6, 2024

Contents

1 Preface

The following document contains justifications annd explainations 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

Recognizing that some images are large and are difficult to see, .png versions have been uploaded to the GitHub repository for this project for convenient viewing.

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 losely followed due to time constraints. For more information on which corners were cut, see Section 6.

3 Development Plan

Development of Bookdex will span 5 weeks. A visual layout of the development plan is shown in Figure 1.

To summarize the figure, there are several tasks that must be completed and each are allotted only so much time over the 5 weeks. The design document, this document, is the only task to span all 5 weeks as it will be updated as updates to other tasks occur.

In the first week and part of the second week, much of the planning will occur. This includes diving deeper into TDS and constructing useful diagrams to better understand the required interactions between the client and server and their various states. A UML class diagram for the TDS protocol will be created so the core of the project can be observed at a glance. An additional UML class diagram will be created for the overall Bookdex application which will make use of the TDS implementation. As required, a preliminary project report will be written and submitted at the end of the first week to ensure the final goal is attainable.

As the planning stage comes to a close in week 2, TDS implementation will begin. This will be the largest part of Bookdex but the time spent planning should accelerate the implementation. To avoid becoming bored of implementing TDS, both the client GUI and server database creation will occur in parallel. It is expected that this implementation phase will take roughly a week and a half to acheive a testable prototype.

The remaining 2 weeks will be spent on testing and bug fixing. Of course, as issues arrise in the initial implementaiton, they will be addressed. These 2 weeks will be used to determine the limits of Bookdex.

It is not included in Figure 1, but, if time permits, additional ideas outlined in the project proposal (visible in the project's Github repository) will be added.

At the end of the 5 weeks, Bookdex will/should be a functioning application.

4 Classes

5 Interactions

6 Decisions

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

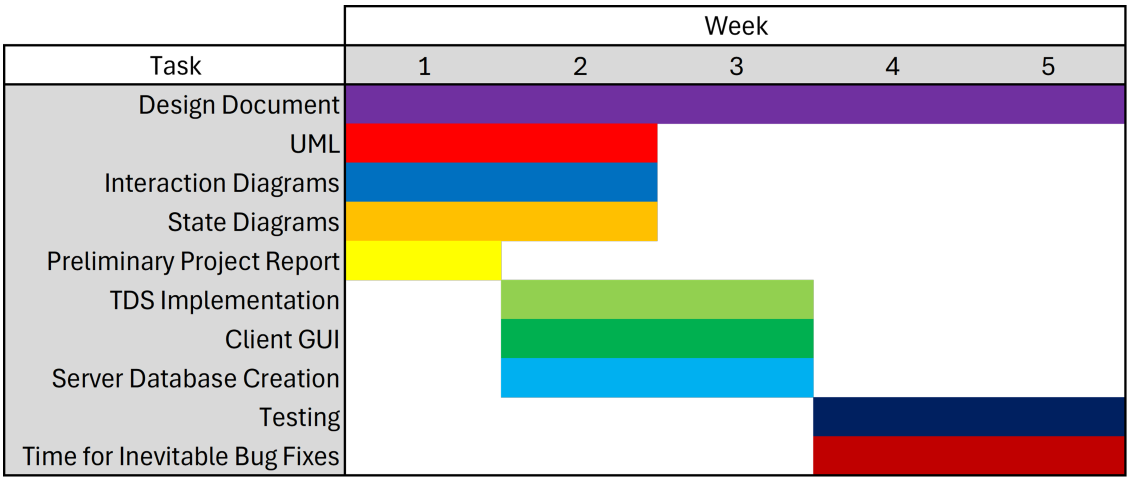

Figure 1: Gantt chart outline for the development plan for Bookdex.

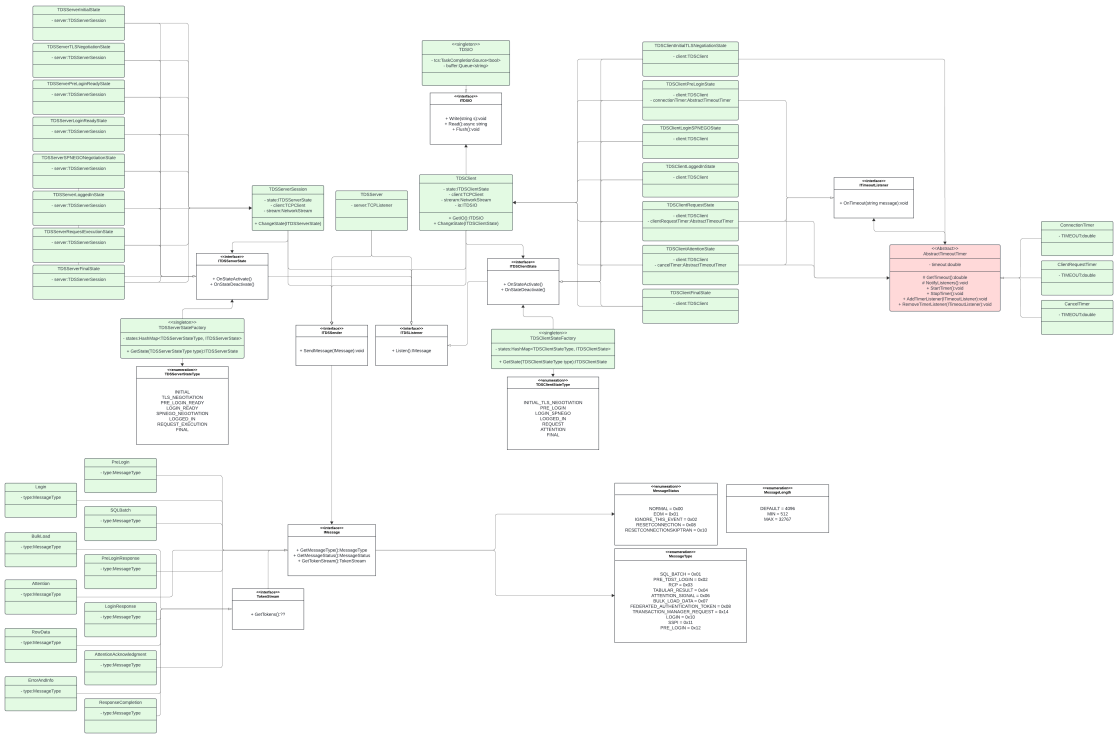

Figure 2: UML diagram outlining the structure of the TDS protocol implementation.

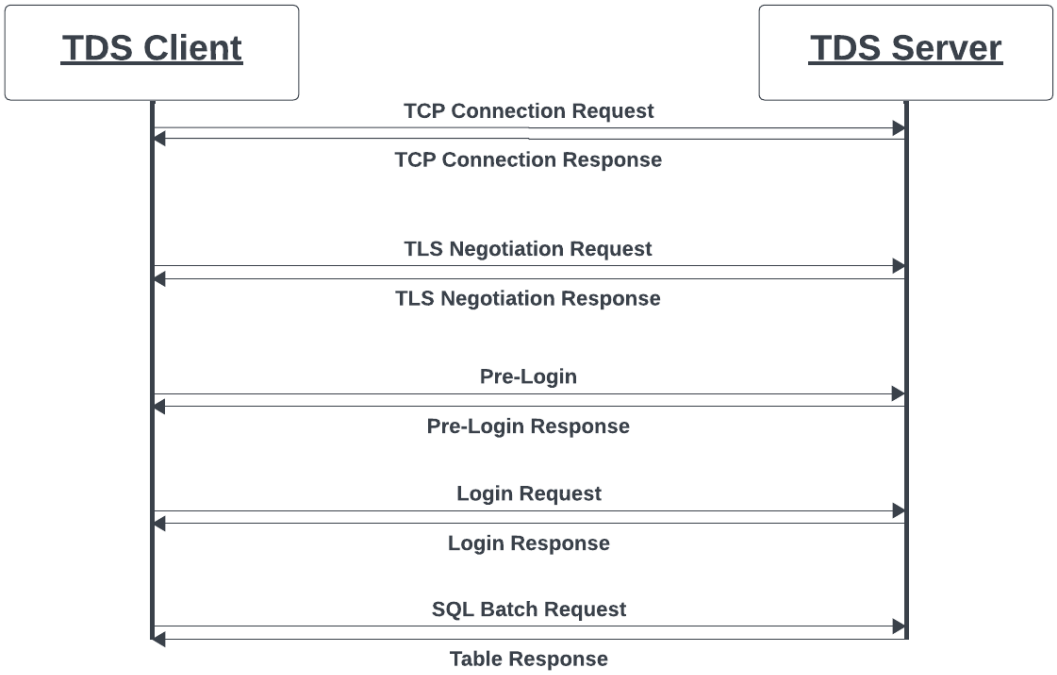

Figure 3: A simplified view of how a client-server interaction unfolds.

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 seperate its implementation from the main Bookdex implementation. This is visible in the project repository as the "bookdex" directory is seperate 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 TDSClientLoggedInState, 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 TDSClientLoggedInState. The queue is then emptied and the TDSClientLoggedInState can do with the inputs what it will. This way, implementation of a view, whether GUI or console, will be supported (I think).

7 Retrospectives

7.1 Week 1

This week began with work on the UML class diagram for the TDS protocol. During its creatation, 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.