**ECE297 - Communication and Design**
**Milestone 3 - Design Document**
**Team #: CD-074**
**Course Instructor: Ted Nolan**
**Shubham Manchanda - 999812225**
**Harsh Verma - 998734482**
**Mandheer Khabra - 999865374**
**Date Submitted: March 9, 2014**

**Revisions made:**

Introduction:

- Some sentences are written with greater concision and more appropriate diction is used

Software architecture:

- Modified the UML diagrams to better depict the sequence of actions using different coloured uni-directional arrows.
- Cut down on the explanation of the UML sequence diagram. instead, laid more emphasis on how to read and understand the diagrams better.

System Requirements

- Secondary functions added
- Additional research conducted for objectives, however the content remains unchanged
- Additional and missed constraints also added

Design Decisions

- Reduced the background info for “Choosing a Shell” and “Choosing a Data Structure”
- Moved the table in “Choosing a Data Structure” to Appendix E
- Included a table of errors for “Dealing with Server/Client Failures” in Appendix D
- Moved weighted decision matrix in “Choosing a Data Structure” to Appendix G
- Removed “Choosing a Communication Protocol”

Conclusion

- What comes next: the next immediate step as well the long term plan is added

**Executive Summary**

With the rising demand for information, effective management of large sizes and varying types of information has become imperative. In addition to that, the growth of the mobile world has shifted so that on-the-move access to information seems like a necessity, not a luxury.

Our client, Cineplex Entertainment, is a business that provides a variety of products and services. The most popular service they provide is screening movies in their theatres. A new opportunity is in place for Cineplex to expand its customer base with the ever-growing demand for information, cell phone and internet use. In response to this, our design proposes a storage system that can manage both large quantities and a variety of information without compromising the faster access time.

This document identifies the problems which movie-goers may have often experienced. Furthermore, the document goes on to elaborate on the system requirements with regards to database operations such as inserting, retrieving, querying data. It also specifies the restrictions that must be considered while performing the above stated operations and the most important objectives and the basis of their importance.

Keeping the system requirements in mind, a technical description of the software architecture has been laid out which clearly highlights the main components as well as the sequence in which some of the basic operations are invoked. Since the primary functionality of the storage system is to retrieve information for the customer, the design team was able to develop two use-case scenarios: purchase of tickets and browsing for movie information. An analysis for both cases has been presented accordingly identifying the stakeholders and their interests, the preconditions, the minimal and success guarantees.

Two important decisions we made were choosing a data structure and discussing code compatibility. Since our aim is to manage both a large quantity and variety of information, we chose the hash table with chaining as our data structure because it excelled at retrieving information to be accessed by the user. We also wanted to create a system that would be forward compatible, meaning it had the ability to be adapt to the changing demands of information in the future.

At present, the storage system is limited to single user access at a time. The next step forward would be to add a multi-threading functionality to the system and ultimately the data structure will have to be configured to support parallel processing thus maintaining fast access speeds.

**TABLE OF CONTENTS**


**1.0 Introduction** 6

*1.1 Problem Statement*

**2.0 Software Architecture** 7

*2.1 UML Component Diagram*

*2.2 UML Sequence Diagram*

**3.0 System Requirements** 8

*3.1 Functions*

*3.1.1 Primary Functions*

*3.1.2 Secondary Functions*

*3.1.3 Unintended Functions*

*3.2 Objectives*

*3.3 Constraints*

**4.0 Use Case Scenarios** 10

*4.0.1 Primary Actor*

*4.0.2 Stakeholders and Interests*

*4.1 Scenario 1*

*4.1.1 Level*

*4.1.2 Stakeholders and Interests*

*4.1.3 Preconditions*

*4.1.4 Minimal Guarantee*

*4.1.5 Success Guarantee*

*4.1.6 Main Success Scenario*

*4.2 Scenario 2*

*4.2.1 Levels*

*4.2.2 Preconditions*

*4.2.3 Minimal Guarantee*

*4.2.4 Success Guarantee*

*4.2.5 Main Success Scenario*

**5.0 Design Decisions** 14

*5.1 Choosing a Shell*

*5.2 Dealing with Server/Client Failures*

*5.3 Choosing a Data Structure*

*5.4 Code Compatibility*

*5.5 Parsing the configuration file*

*5.6 Communication between the query function and the server*

*5.7 Unit test cases*

**6.0 Conclusion** 16

**7.0 List of References** 17

**8.0 Appendices** 18

## 1.0 Introduction

Author : Shubham Manchanda(999812225) Editing: Harsh Verma(998734482)
Mandheer Khabra(999865374)

Our client, Cineplex Entertainment, is one of the largest entertainment companies in Canada[1]. The primary entertainment service that it provides are its movie theatres for watching particularly newly released films[1]. Our client aims to attract more customers by making more information accessible and thereby increasing customer satisfaction. This document will provide a method for Cineplex to increase customer satisfaction.

### 1.1 Problem Statement

In this current era, the use of smartphone devices and the internet is consistently increasing and individuals are becoming increasingly reliant on technology[2]; The demand for information is constantly increasing, as people want to find information faster and from where they are; many individuals don't have the time for planning and the limited amount of information that they can attain from their mobile devices makes it difficult for them to plan on the spot[1].

Today's movie-goers have a complication associated with this. Movie-goers have a desire to attain comfortable and enjoyable seats but unfortunately many of these movie-goers end up dissatisfied with an uncomfortable seat due to a lack of information that indicates the number of seats there are in a theatre[1]. In addition to this, people also like to know the length of trailers and advertisements prior to the film's screening when they are getting late for a movie and unfortunately there is no public database that encompasses this information exists either[1]. Lastly, besides titles such as DBOX, (advertised as vibrated seating) UltraAVX and IMAX (advertised as theatres with larger screens, more comfortable seating and better sound quality), there is very limited online information in regards to each theatre's seating, sound and projection quality, and the size of the screen; there are many instances in which many movie-goers do not expect an enjoyable experience in regards to a theatre[1].

Although movie-goers can now purchase tickets off of their smartphones, Cineplex Entertainment has not optimized the satisfaction of its customers with the growing number of smartphone users. With this new opportunity, Cineplex has the potential to increase its sales and have more satisfied customers.

Therefore, our design team brainstormed the idea of establishing a database that serves the customers of cineplex by providing a greater variety of relevant information. This database would be integrated into the current existing mobile application that Cineplex has for its customers, as well as, online on its website. This includes theatre and screen size, the availability of seats in terms of rows and columns, and the length of advertisements and trailers prior to the actual screening.

## 2.0 Software Architecture

Author : Harsh Verma(998734482) Editing: Mandheer Khabra(999865374), Shubham Manchanda(999812225)

We have achieved the abstraction of function calls by breaking down our design into five important components (Figure 2.1): the *Client Application Interface*, the *Client Library*(storage), the *Utilities*, the *TCP/IP socket layer*, and the *Server*. In Figure 2.1 the **solid brown** arrow depicts the transfer of data from the client to the server while the **dotted brown** arrow depicts the response from the server to the client.

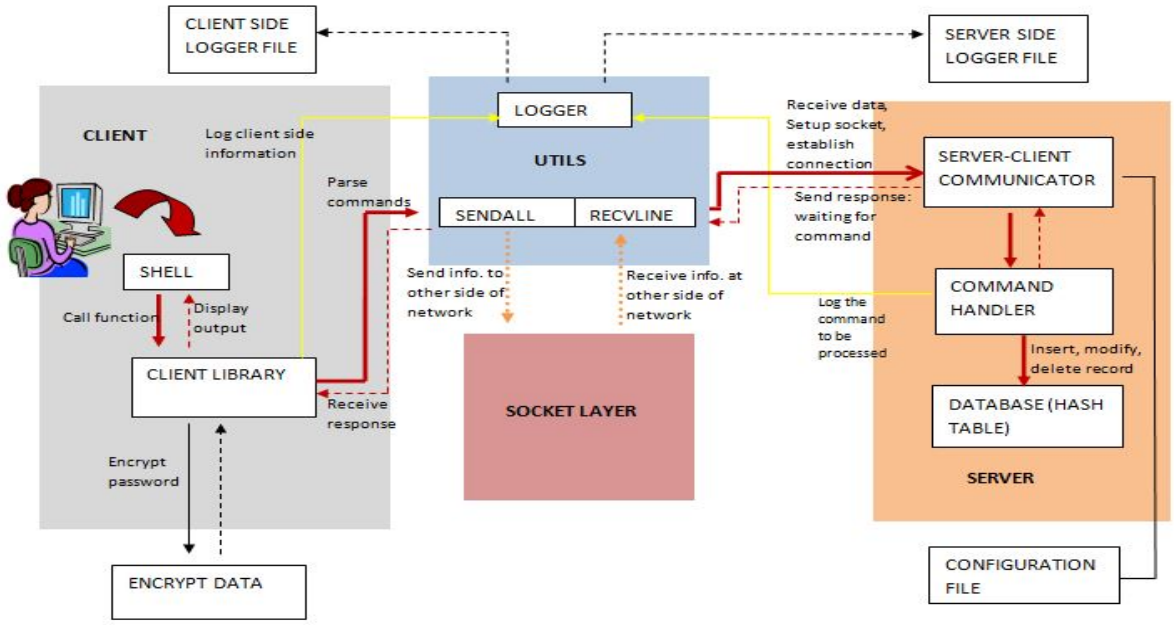


Figure 2.1: UML Component Diagram

- Client Application Interface:
  - the shell takes the commands to be processed from the user, calls the required function from the client library, and finally displays the results to the user.
- Client library:
  - depending on the input command, the required function validates the input information format and sends it to the server
- Utilities:
  - *logging*: writes relevant information into the client side and the server side log files. Helps tackle unforeseen problems.
  - *sendall and recvline:* facilitates the sending and receiving of information by putting the data into a buffer and calling the required functions from the socket layer
- TCP/IP Socket layer [3]:
  - using its functions to exchange data between the server and utils in a sequential order(stream) rather than all of it at the same time
- Server:

- Implements a storage database. Communicates with the client via utils(calls TCP). Reads information from the configuration file to authenticate user, and is assisted by a command handler which makes the changes to the database.

Having already discussed the mode and method of data exchange above, we will now elaborate on the sequence(Figure 2.2) of the calls in the client-server interaction.

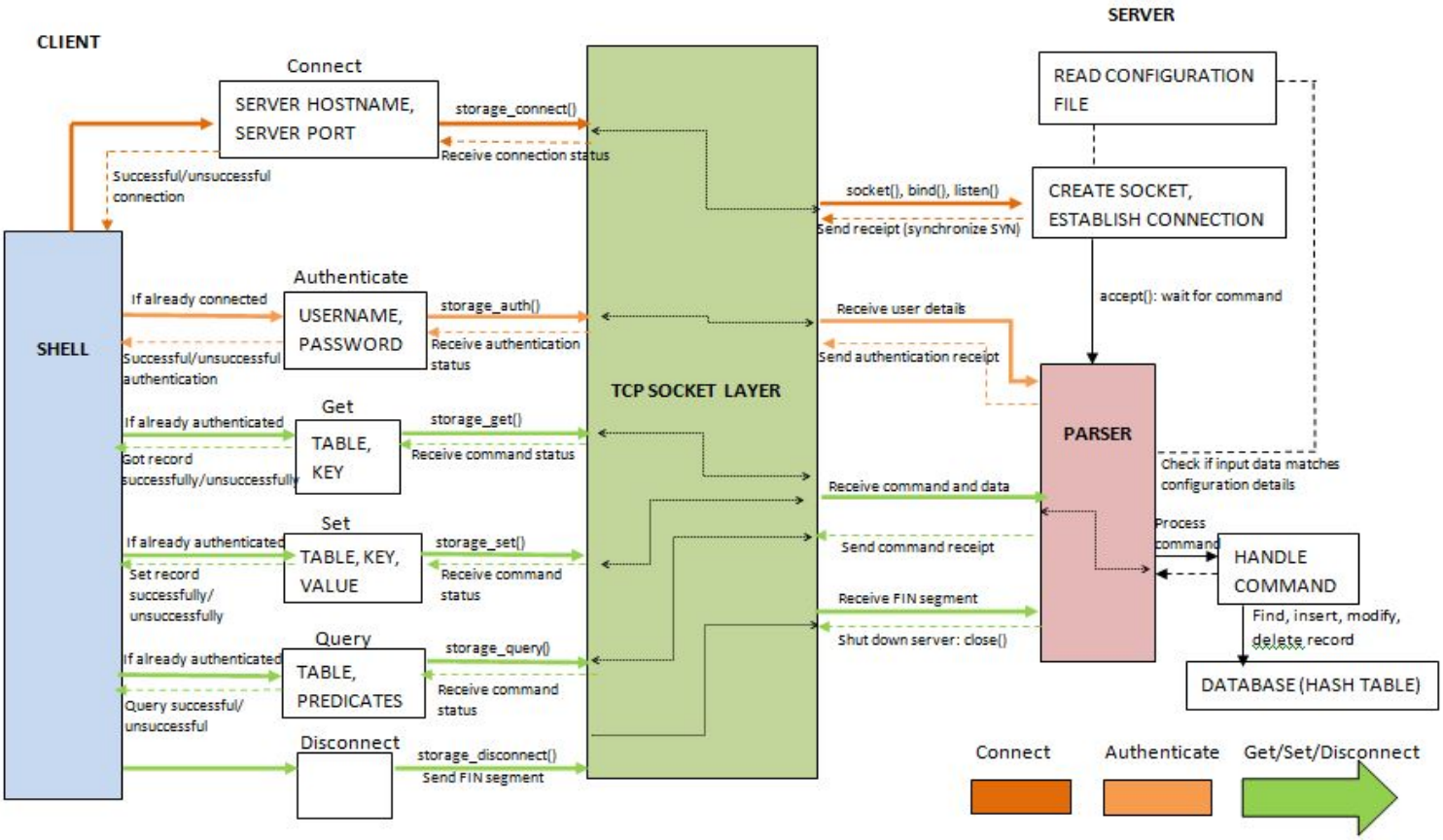


Figure 2.2: UML sequence diagram depicting the order of client-server communication

- The **solid arrows** depict the communication of information from the client side to the server side and the **dotted arrows** depict the response from the server side to the client side.
- The **three** different colours of the arrows in the sequence diagram(Figure 2.2) identify the order of the commands. i.e Step 1. a connection with the server must first be established(orange), Step 2. the user must be validated(peach), Step 3. Following the above stated two steps the user may perform get,set,disconnect operations(green).

## 3.0 System Requirements

Author : Shubham Manchanda(999812225) Editing: Mandheer Khabra(999865374), Harsh Verma(998734482)

The following section outlines the functions, objectives and constraints of the design.

### 3.1 Functions

The fundamental functionality of this project is to store and present information in a form of a

definite article. The types of information that will be presented includes theatre specifications, current seat availability and length of trailers and advertisements prior to screening.

3.1.1 Primary Functions

The design will perform the following fundamental tasks:

- Store and present theatre specifications, current seat availability and the length of trailers and advertisements

3.1.2 Secondary Functions

The following functions simplify, enable, or result from the primary functions:

- Increase the sales of Cineplex Entertainment
- Increase Cineplex's customer satisfaction
- Reduce the number of unoccupied seats within theatres
- Decrease the profits of competing companies

3.1.3 Unintended Functions

The following functions result as a product of the primary functions and the execution of the design:

- Impact the sales of advertised movies prior to the screening a film
- Alter the appearance of cineplex's virtual environment
- Influence the punctuality of cineplex's customers

**3.2 Objectives**

The following section lists what the design should be. Reasons for why these objectives were selected is shown through the evidence in research. The sub-bullets in this section are objective goals. The objectives are listed in order of importance as ranked in Appendix A.

- *Reliable* - The database should display accurate information[4][5]
    - Theatre specifications and advertisement information on all cineplex movie theatres should always be displayed
    - A supporting software should be installed that automatically loads the current seat availability on to sources where this database's information is displayed
- *Accessible* - The database should be easy to access for customers[6]
    - The database should be available online on cineplex's website by the use of a computer or mobile website and smartphone application, as, well as, an automated answering machine if a customer calls for the information in this database
- *Easy to Maintain* - Information must be easy to insert into the database[7]
    - An employee with an average typing speed (Approximately 40 words per minute [8]) should be able to insert any kind of information regarding a single movie theatre in no more than a minute
- *Fast* - The information should cause minimal delay in addition with cineplex's existing virtual environment[9]
    - No more than 0.5 seconds (expected driving perception time[10]) of delay should occur with the addition of this database
- *Aesthetically appealing* - The database should be displayed in a user-friendly and easy-to-read manner[11]

- An adult with average reading speed (250 words per minute [12]) should be able to read the information that they desire within 20 seconds

### 3.3 Constraints

This section outlines requirements the design must meet.

- The storage.h file of the code must remain unchanged.
- The length of all parameters must abide by the lengths described in the given storage.h file; these parameters may be found under Appendix H
- A limit of one hundred tables and one thousand records may be applied at a single period of time
- All coded files must be written in C
- All information must be retrievable with the use of a query search function

## 4.0 Use Case Scenarios

*4.0.1 Primary Actor:*

The functionality of our system is designed to allow customers to access more information. Therefore, the primary actors for this system are the customers of Cineplex Entertainment.

*4.0.2 Stakeholders and Interests:*

Stakeholders include any group of people who may be affected by the project. A stakeholder's interest is defined as the aspect of the stakeholder which will be affected by the design. Interest implies that there may be a gain or loss to the specific stakeholder. The following table highlights the stakeholders and their interests that are common to both the use case scenarios:

| **Stakeholder** | **Interest** | **Primary influence on System Requirements or Cineplex Entertainment** |
|---|---|---|
| Cineplex customers | Concerned with the value, quality, and costs associated with products and services provided by Cineplex Entertainment | ● *Secondary Function:* The database will increase customer satisfaction |
| Cineplex Entertainment, respective Board of Directors and other shareholders | Concerned with making the maximum possible profit within Cineplex Entertainment | ● *Secondary Function:* The database will increase revenues for the company |
| Movie creators - Includes actors, directors, producers, etc. | Interested in the promotion of their created film(s) and also interested in increasing their own profits through endorsements and advertisements | ● *Secondary Function:* Endorsements and advertisements may increase profits for Cineplex or affiliated products and/or services |

| | | |
|---|---|---|
| Other advertisers | Interested in the promotion of their products and/or services | ● May influence profits of affiliated products and/or services |
| Other entertainment competitors (AMC, Cinemark, etc.) [13] | Concerned with impact on size of customer base | ● *Secondary Function:* Service changes may give Cineplex an increased customer base over the competitors |

**4.1 Scenario 1:** Our first scenario is regarding customers who have made the decision to watch a movie and are in the process of purchasing tickets.

*4.1.1 Level:*

Since the system benefits both the customers and Cineplex Entertainment, in this case, it would be on a summary level. Cineplex would benefit because the transaction would increase their revenues while the customers would benefit because they have the ability to access information that is better suited to their needs.

*4.1.2 Stakeholders and Interests:*

In addition to the stakeholders identified in section 4.0.2, we found some unique stakeholders that only concern the first scenario:

| **Stakeholder** | **Interest** | **Primary influence on System Requirements or Cineplex Entertainment** |
|---|---|---|
| Police agencies and organizations against driving while using mobile phones | Concerned with the well-being and safety of drivers and pedestrians | ● *Unintended function:* New database may impact the use of mobile phones while driving |
| Box office attendants | Concerned with their job security from increased online ticket sales | ● *Unintended function:* Online ticket sales may lead to job cuts |

*4.1.3 Preconditions:*

In order for the system to function in regards to this scenario, the customer must have the following information readily available before proceeding to purchase tickets: name of the movie, name of the theatre, screening type, a show time that has not already started, number of tickets to purchase, and payment information.

*4.1.4 Minimal Guarantee:*

To ensure that the interests of the customers are protected, the system is designed to send feedback to a customer regarding the status of their transaction. This feedback will be sent regardless of whether or not the transaction was successful. The feedback may be an email

regarding a successful transaction with a printout of the tickets attached or a pop-up message after submitting a transaction to provide immediate feedback if an error occured.

*4.1.5 Success Guarantee:*

In this first scenario, Cineplex's objectives are met when a confirmation for the payment has been received from the online transaction organization. On successfully making the payment for the movie tickets, the customer will receive a digital receipt along with the tickets. In case of a failed transaction, an error message will be displayed on the client application. For both transaction outcomes, the customer's financial interests are taken care of by relaying the status of the transaction.

*4.1.6 Main Success Scenario:*

1. Customer selects a movie from a drop down list of movies that are "now playing".
2. Customer selects a theatre from a list of the closest theatres to their current location.
3. Customer selects the screening type (CC, UltraAVX, 3D, IMAX, D-BOX, etc.)
4. Customer selects a showtime that has not yet already started.
5. Customer indicates the number of tickets to be purchased.
    a. Type of tickets don't comply with motion picture rating system (G, PG, 18A, etc.)
    b. Number of tickets cannot be processed in the same section of the theatre.
    c. Number of tickets exceeds ticket availability.
6. Customer inputs payment information.
    a. Customer redeems Scene points or vouchers/coupons to pay for tickets.
    b. Invalid payment information inputted.
7. System provides digital receipt to email provided.
    a. Error occurs when processing transaction.

**4.2 Scenario 2:** Our second scenario is regarding customers who are interested in watching a movie, but have not made a final decision yet and are still trying to access information about a movie, a theatre, or ticket prices.

*4.2.1 Level:*

Customers may have the desire to attain different forms of information and thus a variety of customers seeking different forms of information. Therefore, the system benefits this variety of customers and thus the system is also on the summary level.

*4.2.2 Preconditions:*

In this case, no specific preconditions must exist as customers will only use an open search engine. Customers may, however, have an idea of what movie they wish to watch and where they wish to watch it. At least one parameter must be known for the search engine to function, whether that be a movie name or theatre location.

*4.2.3 Minimal Guarantee:*

To ensure that the interests of the customers are protected, the system is designed to provide results to any search query regardless of the information that is trying to be attained. The different kinds of information that could be searched includes: movie descriptions, theatre descriptions, trailers, reviews, ticket prices, screening type descriptions, and seat/ticket availability. Depending on the results of the query, the system could either provide methods to filter results for a more concentrated search or it could provide recommendations for customers who are still unsure of what their next steps should be.

*4.2.4 Success Guarantee:*

In this case, the customer's objectives are met when the search query has been completed and the customer is satisfied with the information obtained through their search query. This could be determined when the customer exits the search query or enters the process for purchasing tickets.

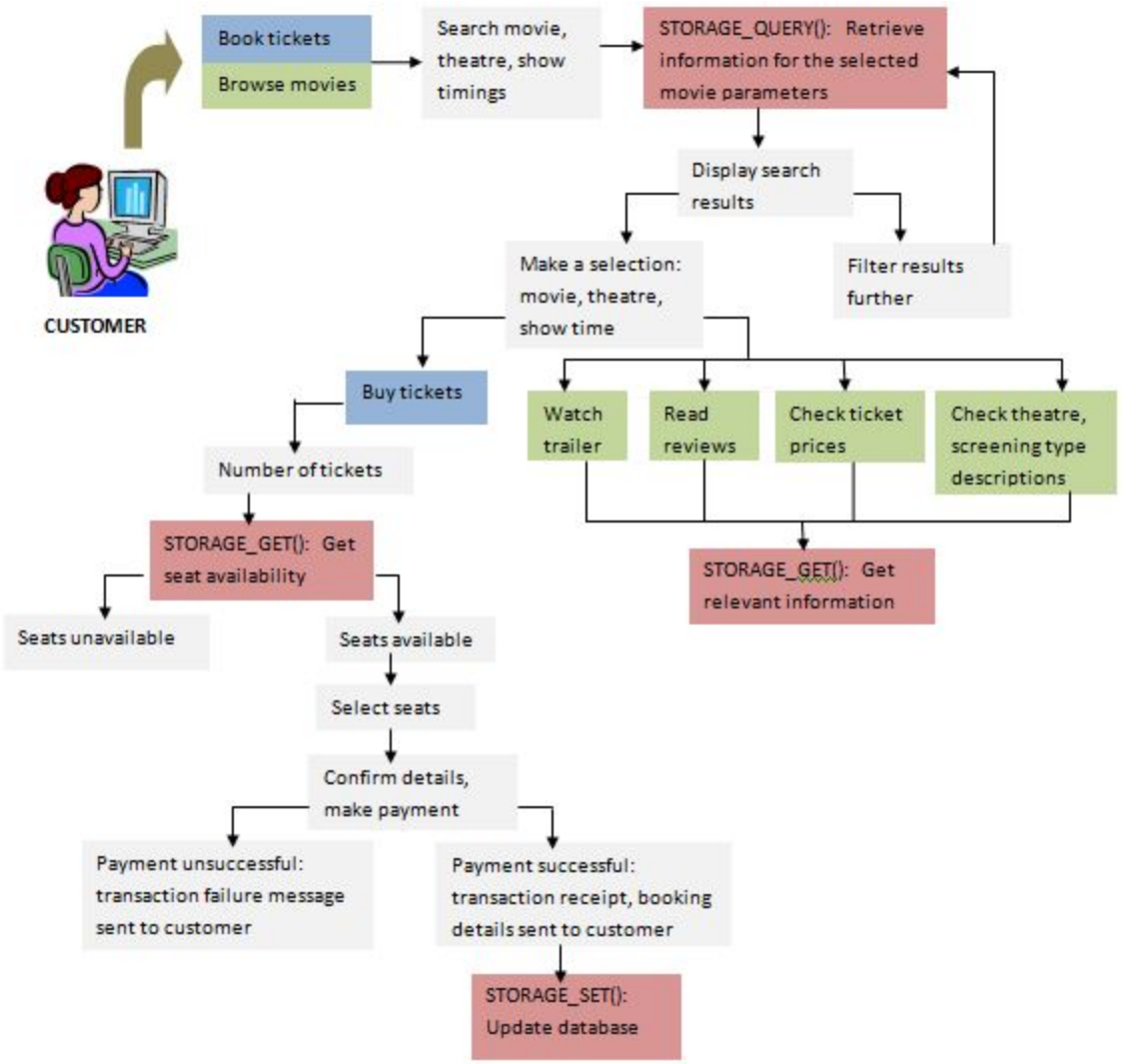


Figure 4: Use case diagram for scenario 1 and 2

*4.2.5 Main Success Scenario:*

1. Customer enters any movie or/and theatre parameters in a general search box.
2. Customer selects the most relevant search result
    a. Watch the trailer - movie parameter is entered
    b. Check reviews - movie parameter is entered
    c. Check ticket prices - theatre parameters are entered
    d. Check screening type descriptions (CC, UltraAVX, 3D, IMAX, D-BOX, etc.) - theatre parameter is entered
    e. Check theatre descriptions (theatre size, theatre capacity, screen dimensions) - theatre parameter is entered
    f. Proceed to purchase tickets - movie and theatre parameters are entered
3. Customer may repeat process to continue browsing or add additional parameters to their search in the general search box to obtain more specific search results

## 5.0 Design Decisions

Author : Mandheer Khabra(999865374) Editing: Harsh Verma(998734482), Shubham Manchanda(999812225)

### 5.1 Choosing a Shell

We had to decide which one of our shells would be used for the remainder of the semester. We decided to run each of the shells against test cases that we had come up with during the first milestone. We also ranked the shells on the coding style which included parameters like formatting, use of variable names, and quality of logging. Using timestamps we also tested the processing time to see which shell took the longest to run. We decided to use a piecewise comparison chart (Appendix B) to rank the importance of each of these parameters and then a weighted decision matrix (Appendix C) to decide which one of the shells would best suit our needs according to the parameters.

We decided to use Harsh's shell based on the formatting of the code, how easily it could be understood by anyone trying to read the code, and its ability to pass every test case. The shell was able to pass all of the test cases from the auto-tester in the first milestone and the corner cases that we came up with as a group. The commenting on the code was also very well detailed such that any unfamiliar code was given a small description as to how it functioned.

### 5.2 Dealing with Server/Client Failures

Most of the failures result from errors produced by the inputs to the server/client, but with our client library we've created a parsing filter that insures that the server is receiving valid information before any functions are called. For connect(), authenticate(), and get() the parsing filter validates input in the client library, while for query() and set() some of the validation will have to be done on the server side. In case of an unsuccessful operation, an error message (Appendix D) is displayed. To allow the client to understand where the error occurred, our error messages also include exactly which piece of information is causing the error.

### 5.3 Choosing a Data Structure

The main decision was deciding what kind of data structure we wanted to use for storing our records and tables in the main memory. We narrowed it down to three choices: an array of records, a hash table for each table, and linked lists. Since a majority of the assignment had to do with “getting” and “setting” values for records in tables, we decided we would test each of the data structures against the difficulty of performing the various processes needed in the second milestone. Appendix E shows the various tests we ran against each of the structures.
As more information is added to the data structure, it’s speed and accessibility would suffer because it gets harder to traverse the data. We decided to go with the hash table with chaining as it performed considerably better than the other data structures. The comparison results for why we chose the hash table can be viewed in Appendix G and F.

**5.4 Code Compatibility**

After thoroughly going over the new requirements, we identified the portions of code that needed to modify, as well as the portions that didn’t require any changes from the previous version of the code:

- Configuration file parser: the configuration file would have to be parsed differently as now along with the table names the columns in the table, their data type and size were also defined. It meant that in case of a query() or a set() call, our code would need to need to validate the input according to the given table properties in the configuration file
- Set(): the set() call is not backward compatible as the format of input has changed for the case when the user wants to insert or modify a record in a table. The input for an older version of the system would give an error, as then the user didn’t have to specify the columns while making set calls.
- the connect(), authenticate(), get() function definitions are backward compatible and don’t need to modified to satisfy the new requirements.

**5.5 Parsing the configuration file:**

There is a need to develop a parser which effectively extracts/parses the configuration parameters from the configuration file. After much deliberation, we identified ease of code modification as the most important consideration in our decision making and decided on using lex to parse the configuration parameters from the configuration file.

The representation of the configuration parameters and their types in the configuration file will change over time. This would require a significant amount of modification to the parsing code in the case of a custom/manual parser as conflicts may arise among certain parsing techniques and data representations. On the other hand, the lex tokenizer utility allows us to add additional tokenizing parameters and methods quite effectively without making wholesale changes to the parser. Also, the lex tokenizing technique is widely practised in the industry and would allow new members of the development team to understand the parsing implementation quicker and make maximum use of its functionality.

**5.6 Communication between the query function and the server**

For querying information, the user input will be formatted as a string of comma separated predicate. The predicate consists of a column name, an operator, and a value.The predicates string passed to the client library will look like:

```
"marks > 50, course = ece297"
```

The above predicate along with the table name will be be sent from the client library to the server where the parser (using lex) will extract the query parameters and validate the parameters. An error message will be displayed in case the query parameters are invalid. If the parameters are valid, then a search will be performed on the database for the entries which hold true for the given query, and a string of the keys corresponding to the query-satisfying entries will be created. To make sure that the number of matching keys did not exceed the maximum number of keys, we needed to count them. And to do that, we had to separate the keys with a separator character (';' or ',', etc). Following the convention of having comma separators in configuration file, we decided on separating the list of matching keys with a comma separator which would make counting the number of keys (tokens) using lex tokenizer extremely convenient.

### 5.7 Unit test cases

As mentioned earlier, the three major changes/additions to the code comprise of changing the configuration file parsing mechanism, allowing the user to perform a query, and changing the input validation process for set() calls. Keeping the above in mind, we felt that in order to ensure proper functioning of the system the code must be able to handle the following error situations:

1. Input configuration file: missing/duplicate parameters, invalid column type, invalid column size, invalid formatting
2. User input for query(): invalid columns, invalid value of column, missing/invalid operator, invalid formatting of predicates, the list of query-satisfying keys exceeds the maximum number of keys allowed
3. User input for set(): invalid order of columns, invalid column data type, missing column/value

## 6.0 Conclusion

Author: Mandheer Khabra(999865374) Editing: Harsh Verma(998734482), Shubham Manchanda(999812225)

In today's world of growing internet and mobile use, Cineplex Entertainment has a strong opportunity to raise customer satisfaction and increase profits. The design outlined in this document integrates a storage database into the Cineplex's virtual environment with faster access to information. Keeping in mind, that with time there will be increase in the amount of data to be dealt with, our design provides room for scalability to maintain faster transactions and data exchange. Having done some research, the design team has identified the two phase hashing process as a possible solution to making our hash table compatible for parallel processing of data[14]. However, further development of the hashing algorithm is required. Work also needs to be done towards extending the storage functionality to multiple users at a time.

## 7.0 List of References


[1] Guest Services, Cineplex Entertainment. [Phone]. Accessed February 5, 2014, Available: 1-800-333-0061

[2] Internet, broadband, and cell phone statistics, Lee Rainie. [Online]. Accessed February 5, 2014. Available: http://www.distributedworkplace.com/DW/Research/Internet%20broadband%20and%20cell%20phone%20statistics%20-%20Pew%20Internet%20Report%20Jan%202010.pdf

[3] Socket Programming. *ECE297* [Online]. Accessed February 5, 2014. Available: https://sites.google.com/a/msrg.utoronto.ca/ece297/tas-and-labs/socket-programming

[4] Tech Talks ~ The Importance of Reliability, Tharanga Jalathge, [Online] Accessed March 3, 2014. Available:http://creately.com/blog/development/tech-talks-the-importance-of-reliability-2/

[5] Why the human element of reliability is so important, Ted Melencheck, [Online]. Accessed March 3, 2014. Available: http://www.reliableplant.com/Read/23615/human-element-reliability

[6] The effect of previous exposure to technology on acceptance and its importance in usability and accessibility engineering, Andreas Holzinger, Gig Searle, Michaela Wernbacher, [Online]. Accessed March 3, 2014. Available: http://link.springer.com/article/10.1007/s10209-010-0212-x#page-2

[7] Designing for maintenance: A Game Theoretic Approach, Gabriel Hernandez, Carolyn Conner Seepersad, Farrokh Mistree, [Online]. Accessed March 3, 2014. Available:

[8] Average Typing Speed (2014), *Learn 2 Type*. [Online]. Accessed February 6, 2014, Available: http://www.learn2type.com/typingtest/typingspeed.cfm

[9] The need for Speed, *Guillermo Rauch,* [Online]. Accessed March 3, 2014. Available: https://cloudup.com/blog/the-need-for-speed

[10] Driver Reaction Time, Marc Green, [Online]. Accessed February 6, 2014. Available: http://www.visualexpert.com/Resources/reactiontime.html
http://www.me.utexas.edu/~ccseepersad/hernandez.seepersad.eng.opt.game.theory.pdf

[11] *The Human Factor*, Kim Vicente, [Book]. Accessed March 4, 2014.

[12] What Is the Average Reading Speed and the Best Rate of Reading?, Mark Thomas. [Online]. Accessed February 6, 2014. Available: http://www.healthguidance.org/entry/13263/1/What-Is-the-Average-Reading-Speed-and-the-Best-Rate-of-Reading.html

[13] Cineplex Inc. Competition, Hoovers (A D&B Company). [Online] Accessed March 9, 2014. Available:http://www.hoovers.com/company-information/cs/competition.Cineplex_Inc.df8c4fe7e6c77b21.html

[14]: Hash table in Massively Parallel Systems, I-Ling Yen and Farokh Bastani [Online] Accessed March 2, 2014. Available:http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00222988&tag=1

## 8.0 Appendices

**Appendix A: Ranking of System Requirement Design Objectives**

This is a copy of the pair-wise comparison used to rank the objectives. All pair-wise comparisons were completed via discussion and subsequently consensus within the team.

| | Aesthetic | Reliable | Fast | Easy to Maintain | Accessible | Score |
|---|---|---|---|---|---|---|
| Aesthetic | X | 0 | 0 | 0 | 0 | 0 |
| Reliable | 1 | X | 1 | 1 | 1 | 4 |
| Fast | 1 | 0 | X | 0 | 0 | 1 |
| Easy to Maintain | 1 | 0 | 1 | X | 0 | 2 |
| Accessible | 1 | 0 | 1 | 1 | X | 3 |

**Appendix B: Ranking of Shell Objectives**

The following pair-wise comparison, which compares shell design objectives, aided in the selection of the shell for our database by easing the weighting process within its weighted decision matrix. A describes how each pair-wise comparison was completed.

| | **Formatting** | **Testing against cases** | **Processing Times** | **Quality of Logging** | **Score** |
|---|---|---|---|---|---|
| **Formatting** | X | 0 | 0 | 1 | 1 |
| **Testing against cases** | 1 | X | 1 | 1 | 3 |
| **Processing Times** | 1 | 0 | X | 1 | 2 |
| **Quality of logging** | 0 | 0 | 0 | X | 0 |

**Appendix C: Weighted Decision Matrix for the selection of Database Shell -** Increase zoom for a better view

| Weighted Decision Matrix for the Selection of Database Shell | | | | | | |
|---|---|---|---|---|---|---|
| **Objective:** | **Weighting** | **Notes:** | | | | |
| Testing against Cases | 40.00% | This weighted decision matrix weighs all the objectives using discussion and consensus within the team | | | | |
| Processing Times | 30.00% | and subsequently it evaluates each individual's database shell with how strongly each shell complies with | | | | |
| Formatting | 20.00% | the weighted objectives in order to select the most appropriate shell. The objectives were rated earlier | | | | |
| Quality of Logging | 10.00% | with a pairwise comparison and can be found under "Ranking of Shell Objectives", which aided in | | | | |
| Total | 100.00% | accurately selecting the weightings. | | | | |
| **Individual:** | **Harsh** | | **Shubham** | | **Mandheer** | |
| **Objective:** | Percent | Weighted | Percent | Weighted | Percent | Weighted |
| Testing against Cases | 100.00% | 40.00% | 85.00% | 34.00% | 90.00% | 36.00% |
| Processing Times | 85.00% | 25.50% | 90.00% | 27.00% | 90.00% | 27.00% |
| Formatting | 85.00% | 17.00% | 90.00% | 18.00% | 85.00% | 17.00% |
| Quality of Logging | 90.00% | 9.00% | 85.00% | 8.50% | 80.00% | 8.00% |
| ***Total*** | | ***91.50%*** | | ***87.50%*** | | ***88.00%*** |

### Appendix D: Table of Error Codes

The following figure represents the error codes used throughout the code to provide feedback to the user when something goes wrong.

```
// Error codes.
#define ERR_INVALID_PARAM 1     ///< A parameter is not valid.
#define ERR_CONNECTION_FAIL 2       ///< Error connecting to server.
#define ERR_NOT_AUTHENTICATED 3     ///< Client not authenticated.
#define ERR_AUTHENTICATION_FAILED 4 ///< Client authentication failed.
#define ERR_TABLE_NOT_FOUND 5       ///< The table does not exist.
#define ERR_KEY_NOT_FOUND 6     ///< The key does not exist.
#define ERR_UNKNOWN 7           ///< Any other error.
#define ERR_TRANSACTION_ABORT 8     ///< Transaction abort error.
```

### Appendix E: Tests Run Against Data Structures

This table represents the tests we ran against each of the data structures where 1 represents the best data structure to perform the indicated process and 3 represents the data structure that experienced the most difficulty while performing the process.

| | **Hash table** | **Linked list** | **Array** |
|---|---|---|---|
| **Inserting records** | 2 | 1 | 3 |
| **Retrieving records** | 3 | 2 | 1 |
| **Modifying records** | 3 | 1 | 2 |
| **Deleting records** | 3 | 1 | 2 |
| **Ease of implementation** | 3 | 2 | 1 |
| **Total** | **14** | **7** | **9** |

## Appendix F: Ranking of Data Structure processes

The following pair-wise comparison compares relevance of the data processes in our database. The description in Appendix A describes how each pair-wise comparison was completed.

| | Inserting records | Retrieving records | Modifying records | Deleting records | Ease of implementation | Score |
|---|---|---|---|---|---|---|
| **Inserting records** | **X** | 0 | 0 | 1 | 0 | **1** |
| **Retrieving records** | 1 | **X** | 1 | 1 | 1 | **4** |
| **Modifying records** | 1 | 0 | **X** | 1 | 1 | **3** |
| **Deleting records** | 0 | 0 | 0 | **X** | 0 | **0** |
| **Ease of implementation** | 1 | 0 | 0 | 1 | **X** | **2** |

## Appendix G: Weighted Decision Matrix for Selecting a Data Structure

| Weighted Decision Matrix for the Selection of the most appropriate Data Structure | | | | | | |
|---|---|---|---|---|---|---|
| **Objective:** | **Weighting** | **Notes:** | | | | |
| Retrieving Records | 33.33% | This weighted decision matrix weighs all the objectives using discussion and consensus | | | | |
| Modifying Records | 26.67% | within the team and subsequently it evaluates each data structure with how strongly | | | | |
| Ease of Implementation | 20.00% | each data structure complies with the weighted objectives in order to select the most | | | | |
| Inserting Records | 13.33% | appropriate data structure. The objectives were rated earlier with a pairwise | | | | |
| Deleting Records | 7% | comparison and can be found under "Ranking of Data Structure Processes", which aided | | | | |
| Total | 100.00% | in selecting the weightings | | | | |
| **Individual:** | **Hash Table** | | **Linked List** | | **Array** | |
| **Objective:** | Percent | Weighted | Percent | Weighted | Percent | Weighted |
| Retrieving Records | 100.00% | 33.33% | 35.00% | 11.67% | 70.00% | 23.33% |
| Modifying Records | 100.00% | 26.67% | 70.00% | 18.67% | 35.00% | 9.33% |
| Ease of Implementation | 35.00% | 7.00% | 70.00% | 14.00% | 100.00% | 20.00% |
| Inserting Records | 70.00% | 9.33% | 100.00% | 13.33% | 35.00% | 4.67% |
| Deleting Records | 100.00% | 6.67% | 35.00% | 2.33% | 70.00% | 4.67% |
| ***Total*** | | ***76.33%*** | | ***57.67%*** | | ***57.33%*** |

## Appendix H: Coding constraints

```
// Configuration constants.
#define MAX_CONFIG_LINE_LEN 1024 ///< Max characters in each config file line.
#define MAX_USERNAME_LEN 64 ///< Max characters of server username.
#define MAX_ENC_PASSWORD_LEN 64 ///< Max characters of server's encrypted password.
#define MAX_HOST_LEN 64     ///< Max characters of server hostname.
#define MAX_PORT_LEN 8      ///< Max characters of server port.
#define MAX_PATH_LEN 256    ///< Max characters of data directory path.

// Storage server constants.
#define MAX_TABLES 100      ///< Max tables supported by the server.
#define MAX_RECORDS_PER_TABLE 1000 ///< Max records per table.
#define MAX_TABLE_LEN 20    ///< Max characters of a table name.
#define MAX_KEY_LEN 20      ///< Max characters of a key name.
#define MAX_CONNECTIONS 10  ///< Max simultaneous client connections.

// Extended storage server constants.
#define MAX_COLUMNS_PER_TABLE 10 ///< Max columns per table.
#define MAX_COLNAME_LEN 20  ///< Max characters of a column name.
#define MAX_STRTYPE_SIZE 40 ///< Max SIZE of string types.
#define MAX_VALUE_LEN 800   ///< Max characters of a value.
```