

# Appointment Management System

## Carpe Diem Massage Company

Fort Hays State University - Department of Computer Science  
Software Engineering Fall 2022

### Group Number 11:

James Chapman - [jachapman3@mail.fhsu.edu](mailto:jachapman3@mail.fhsu.edu)

Landon Crispin - [lccrispin@mail.fhsu.edu](mailto:lccrispin@mail.fhsu.edu)

Daniel Ring - [dmring2@mail.fhsu.edu](mailto:dmring2@mail.fhsu.edu)

### Project URLs:

GitHub - [https://github.com/JamesChapmanNV/CSCI\\_441\\_group\\_11](https://github.com/JamesChapmanNV/CSCI_441_group_11)

Discord - <https://discord.gg/2dXssMwc>

### Submission:

20 November 2022 - Report Three – Final

# Table of Contents

Carpe Diem Massage Company .....	1
Fort Hays State University - Department of Computer Science.....	1
Software Engineering Fall 2022.....	1
Summary of Changes and Distribution of Work.....	4
Customer Statement of Requirements .....	5
Problem Statement .....	5
Glossary of Terms.....	7
System Requirements .....	8
Enumerated Functional Requirements .....	8
Enumerated Non-Functional Requirements .....	9
On-Screen Appearance Requirements.....	10
Functional Requirements.....	11
Stakeholders .....	11
Actors and Goals.....	11
Use Cases.....	12
Casual Description .....	12
Use Case Diagram .....	14
System Sequence Diagrams .....	15
UC-03: Add Masseuse .....	15
UC-06: Check Schedule.....	16
UC-08: Make Appointment.....	17
Traceability Matrix .....	18
Effort Estimation Using Use Case Points .....	19
Project Size Estimation .....	19
TCF .....	19
UCP .....	20
Domain Analysis .....	21
Concept Definitions.....	21
Association Definitions.....	21
Attribute Definitions.....	22
Domain Diagram .....	23
Traceability Matrix.....	23
Interaction Diagrams.....	24

UC – 06 Show Schedule .....	24
UC – 08 Add Appointment .....	25
Class Diagram and Interface Specification .....	26
Class Diagram.....	26
Data Types and Operation Signatures .....	27
System Architecture and System Design .....	33
Class Diagram.....	33
Data Types and Operation Signatures .....	34
Class to Domain Traceability Matrix .....	40
Algorithms and Data Structures.....	41
Algorithms.....	41
Data Structures .....	42
User Interface: Design and Implementation .....	43
User Interface Class Structure.....	44
User Interface Layout.....	45
User Interface Reusability and Modularity .....	48
Design of Tests .....	49
Class Unit Tests .....	49
History of Work .....	53
Current Status .....	53
Future Work .....	53
References.....	54

# Summary of Changes and Distribution of Work

- James Chapman
  - Revised problem statement
  - Updated data structures used
  - Updated algorithm section to validate test designs
  - Updated activity diagram
- Landon Crispin
  - Revisions to data types and operation signatures
  - Revisions to concept definitions
  - Updated user interface class structure
- Dan Ring
  - Document formatting
  - Initial pulling from Reports 1 and 2.
  - Updated Unit Tests
  - Adjusted History of Work and Future Plan of Work

# Customer Statement of Requirements

## Problem Statement

Carpe Diem Massage Company Carpe Diem Massage Company is a small massage company that is expanding to a new building. They will have 4 separate rooms for massages, allowing for increased sales and more flexible scheduling. They need an appointment scheduler for the front desk. When a customer comes in (or calls), the front desk should be able to access available appointment times, collect necessary information from customers, and book appointments. There should be some functionality for the user to choose a preferred masseuse and to search available appointments for a specific masseuse based on the client's needs and desires. It should also record the clients preferred type of massage, such as deep tissue, Swedish, hot stones etc. A masseuse must be able to ask for their schedule to see which room they need to be in, and at what time. For small businesses, customer retention can be an issue, therefore being able to keep track of a clients last massage would make it easier to reach out to those that haven't been in for a while with reminders or promotional deals.

Carpe Diem Massage Company is open Mon - Fri, 9AM - 5PM. Each appointment is 1 hour long and must be scheduled at the top of the hour. Each room will have 8 appointment slots per day (9, 10, 11, 12, 1, 2, 3, 4). Every day will have 32 available appointments. Each appointment slot shall be open or booked. To book an appointment, the front desk employee must assign a masseuse and customer through the appointment scheduler. Payments for the massages are handled between the customer and the masseuse. All massages (1 hour) are charged at a flat rate.

Each masseuse has a schedule of availability, which is decided at time of hire, and the schedule shall not change from week to week. A masseuse can only do one massage at a time, so there must be some record to prevent overlap. Other information to be stored for each masseuse include name, phone number, email, address, license number, employee number, etc.

Each customer should have their basic information collected at the time of their first appointment scheduling. This should include name, phone number, and email.

Potentially, this application should email reminders to customers 24 hours in advance of their appointment time.

Pull, add, remove, update, analyze would all be necessary functions to the scheduler in an appointment app. A search function will allow the front desk employee to quickly access a client's personal information. Each of these functions perform I/O, so a method of record storage will be used in the form of a MySQL database. To allow an end-user to work with the records easily and efficiently, a graphical user interface will be implemented to manage the data with minimal effort. A user

The business would enjoy the ability to add multiple users with different levels of access based on their position within the company. The owner should have Admin status and the ability to perform all functions within the software.

The software should have a pleasing display and easily accessible menus denoting all the necessary features. Minimum interface is required for functionality; however, should time and budget allow it, overall greater UI and software functionality in general would be appreciated.

# Glossary of Terms

**Admin** – The owner of the company and any other so designated by them. Has full access to all software functionalities.

**Appointment** - Agreed-upon time. Should be able to be canceled if need be.

**Booked** - Room, customer, and masseuse cannot be booked at the same time

**Customer/Client** - Person getting massage. Basic information needed.

**Massage type** - Available options for specialized massage.

**Masseuse** - A licensed employee of Carpe diem. Basic information and a weekly available schedule should be collected at time of hire.

**Masseuse availability** - Important job requirement to have same availability each week.

**Open** - Available for appointment to be scheduled at a certain room at a certain time.

**Room** - Labeled 1, 2, 3, 4

**User** – Employee of company with varying levels of access to the system up to and including Admin privileges. Minimal requirements are to make and remove appointments.

# System Requirements

## Enumerated Functional Requirements

Identifier	PW	Description
REQ-01	5	The system shall present an interface to manage appointments and allow the user to manage the appointments.
REQ-02	2	The system shall allow the user to Create, Read, Update, Delete customer information.
REQ-03	4	The system shall allow the user to Create, Read, Update, Delete masseuse information, scheduling, and availability.
REQ-04	4	The system shall interface between the front-end user interface and the cloud hosted SQL database.
REQ-05	2	The system shall allow for the 4 rooms to be booked at the same time, while each room, masseuse, and customer shall NOT be double booked
REQ-06	3	The system shall conduct GUI management and any operations involving data processing in two separate threads to allow for seamless and smooth experience to the end-user without delay.

### Analysis of Enumerated Functional Requirements

The main functional requirement of this application is to allow a user to work with the appointment scheduling database and its appropriate tables (REQ-01). A user should be able to manipulate customer information (REQ-02) and manipulate masseuse availability and information as well (REQ-03). To add flexibility into the scheduling system, the system shall allow the booking of all rooms as required by demand and availability. The software shall not allow overbooking. (REQ-05).

In order to do these things, an interface will allow components of the GUI to interact with the backend database manager (REQ-04). Finally, because the GUI and database management interface are components which require their own processing power, two threads will be incorporated to distribute load and allow for simultaneous processing (REQ-06).



## Enumerated Non-Functional Requirements

Identifier	PW	Description
REQ-07	1	The system should present the data in an easy to use, accessible format.
REQ-08	2	The system should allow appointments to be scheduled using hourly blocks and be restricted to certain periods of time.
REQ-09	1	The system should incorporate a login system which will provide the user access to the software with the appropriate level of permissions.
REQ-10	1	The system should use an admin management system to allow a manager to change certain parameters about the scheduling behavior or allow a manager to change users and permissions.
REQ-11	3	The system should allow appointments to be scheduled only if the time preferred is not already booked.
REQ-12	2	The system should allow the end-user to specify the type of massage to be serviced to the customer and other specifics about the type of work to be performed.
REQ-13	1	The system should provide a way to notify the customer of an upcoming appointment to remove the need for manual notification.

### Analysis of Enumerated Non-Functional Requirements

Requirements listed here are not out of necessity; however, they will significantly aid in the quality of experience in using the software.

For scheduling, the system will allow scheduling to occur only in the blocks that they can be scheduled. It should allow for the changing of the number of rooms available and the store hours of operation in a global manner, (REQ-10) and only if the time to schedule is not already booked (REQ-13). Different UI panels for admin management (REQ-12) and user login (REQ-11) will be incorporated to assign and compartmentalize user access and permissions.

For miscellaneous yet important functionality of the software, an easy to use and accessible format shall be available to the end user (REQ-09). In addition to basic scheduling requirements, the scheduler shall indicate the type of massage/service to be performed on the scheduled time slot (REQ-14).

## On-Screen Appearance Requirements

Identifier	PW	Description
REQ-14	2	The screen shall display separate pages to manage the schedule, customer information, masseuses and their availability in an easy-to-read format.
REQ-15	1	The screen shall be responsive and adapt to various sizes to be able to adjust to multiple different devices as necessary.
REQ-16	4	The screen shall incorporate a nav bar for ease of navigation among different features and functionality.
REQ-17	2	The screen shall display a management interface for a manager/operator to change and adjust parameters and configuration of the schedule to specify how the schedule will operate.

### Analysis of On-Screen Appearance Requirements

The user interface of the software needs to provide functionality for the end-user and provide an appearance in such a way that it is easy to understand and easy to use.

Separate pages/windows will address functionality to each individual component (REQ-16) – a window to manage, view, and edit the schedule, a window to manipulate customer information, and a window to manage, view, and edit masseuses and their availability. A navigational bar will provide a way to access the different windows incorporated into the program (REQ-18).

The screen will be responsive, resizable, and be able to adapt to a number of different screen configurations and operating systems under the premise that Python is in use (REQ-17).

# Functional Requirements

## Stakeholders

**Front desk employee** - The primary user of the appointment management software.

**Masseuse employee** - Person customers get scheduled with.

**Customer** - Person needing to be scheduled with masseuse employee.

**Owner** – Owner of Carpe Diem Massage Company.

## Actors and Goals

**User (Initiating)** – Depending on extra privileges set by Admin, user's goals are to create and manage clients and appointments. Extra privileges include creating and managing masseuses and services as well as setting store hours and appointment length.

**Admin (Initiating)** – Goals are to create and manage users and their privileges. Also to create and manage clients, appointments, masseuses, and services as well as setting store hours and appointment length.

**Database (Participating)** – Goals are to authenticate and connect to SQL database allowing for the transfer and manipulation of data stored there.

**ApptChecker (Initiating)** – Goal is to check the current date against the date of all scheduled appointments and sends an email to the clients with appointments the next day.

**ReminderEmail (Initiating)** – Goal is to schedule email reminder to be sent 24 hours before appointment time. This shall be scheduled at time of scheduling.

# Use Cases

## Casual Description

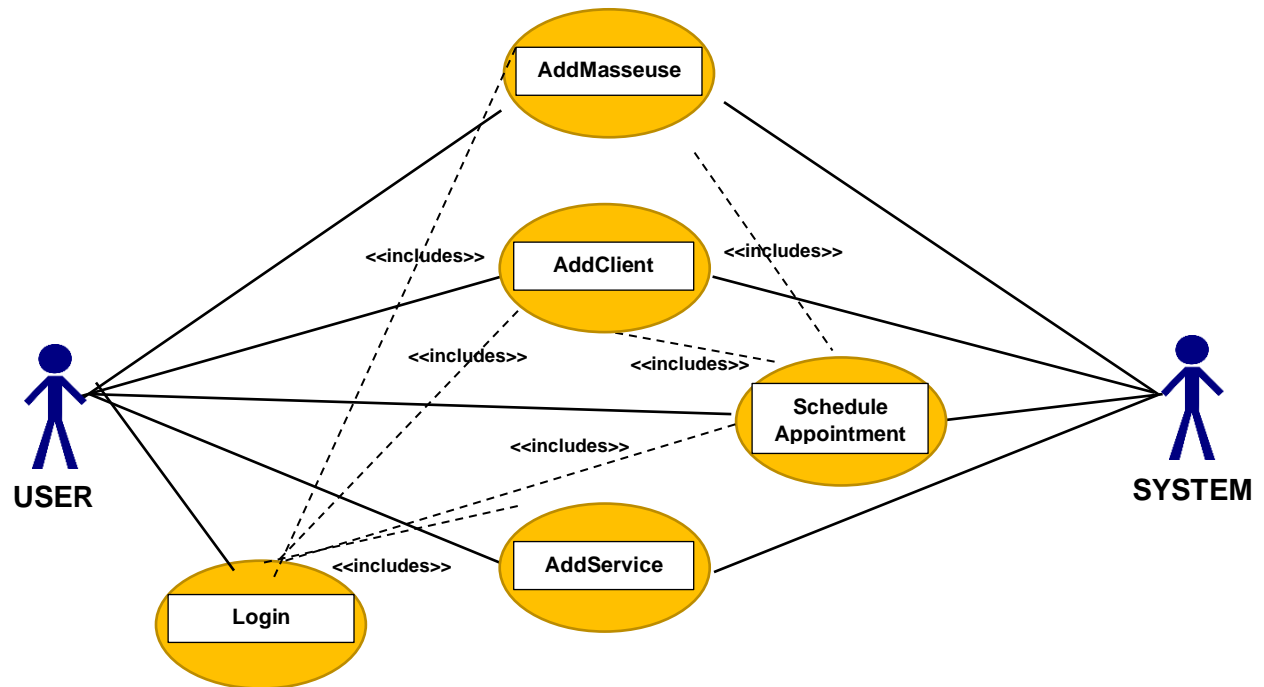
Use Case	Name	Description	Requirement
UC-01	Login	Ensure server connectivity and authenticate user.	REQ-01, REQ-04, REQ-09, REQ-10, REQ-14 - REQ17
UC-02	AddClient	Allows user to add new client information to database.	REQ-02, REQ-04, REQ-14 - REQ17
UC-03	AddMasseuse	Allows user to add new masseuse details.	REQ-03, REQ-04, REQ-14 - REQ17
UC-04	EditClient	Allows user to edit or delete existing client info.	REQ-02, REQ-04, REQ-14 - REQ17
UC-05	EditMasseuse	Allows user to edit or delete existing masseuse info.	REQ-03, REQ-04, REQ-14 - REQ17
UC-06	CheckSchedule	Allows the user to check the scheduled appts on a given day.	REQ-01, REQ-04, REQ-06, REQ- 07, REQ-14 - REQ17
UC-07	GetMasseuseSched	Allows the user to check the scheduled appts of a given masseuse.	REQ-03, REQ-04, REQ-06, REQ- 07, REQ-14 - REQ17
UC-08	MakeAppt	Allows the user to schedule an appointment with desired services and desired masseuse.	REQ-01, REQ-04, REQ-05, REQ-08, REQ-11, REQ-12, REQ-14 - REQ17
UC-09	RemoveAppt	Allows user to delete an appointment.	REQ-01, REQ-04, REQ-05, REQ-14 - REQ17
UC-10	SetandGetShopHours	Allows the user to set and get hours of operation.	REQ-07, REQ-08, REQ-10, REQ-14 - REQ17
UC-11	SetandGetTimeslotLen	Allows the user to set and get the appointment length. (Same for all appointments.)	REQ-07, REQ-08, REQ-10, REQ-14 - REQ17

UC-12	EmailReminder	Emails client 24 hours before appointment.	REQ-13
UC-13	SetandGetServices	Allows the user to set and get the services offered at the location.	REQ-12, REC-14 – REC 17
UC-14	Admin	Super user that is able to add/update/remove users and their privileges.	REQ-04, REQ-09, REQ-10, REC-14 – REC 17

After careful consideration it has been decided that the following Use Cases are non-essential and will possible be incomplete by the end of semester Final Demo as represented by the Future Work section:

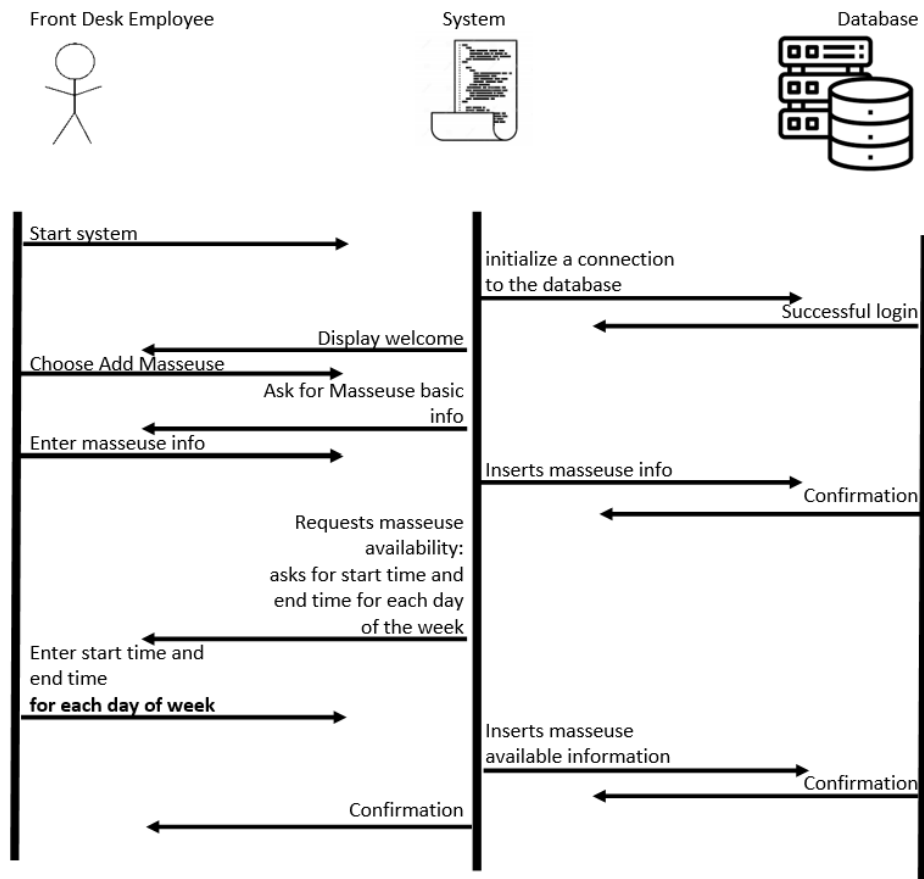
- UC-7 Get Masseuse Schedule – Data will be organized by default by date and time. The Future Work goal is that the data could be organized by Masseuse or even with a separate display indicating a specific Masseuse schedule only.
- UC-10 SetandGetShopHours - This will be set to a default value of 8 one-hour timeslots representing a 9-5pm workday with no business being conducted on Saturdays and Sundays. The Future Work goal is to have this adjustable by the Admin.
- UC-11 SetandGetTimeslotLen – This will be set to a default length of one hour. The Future Work goal would allow Admin to adjust time slot lengths allowing for different times to be allotted to appointments.
- UC-12 Email Reminder – As a default there will be no reminder system implemented. Manual access of customer information from appointments or client list will be necessary for emails to be sent.
- UC-13 SetandGetServices – By default all Masseuses will offer all services that will be negotiated between the Customer and the Masseuse at the time of appointment.

## Use Case Diagram

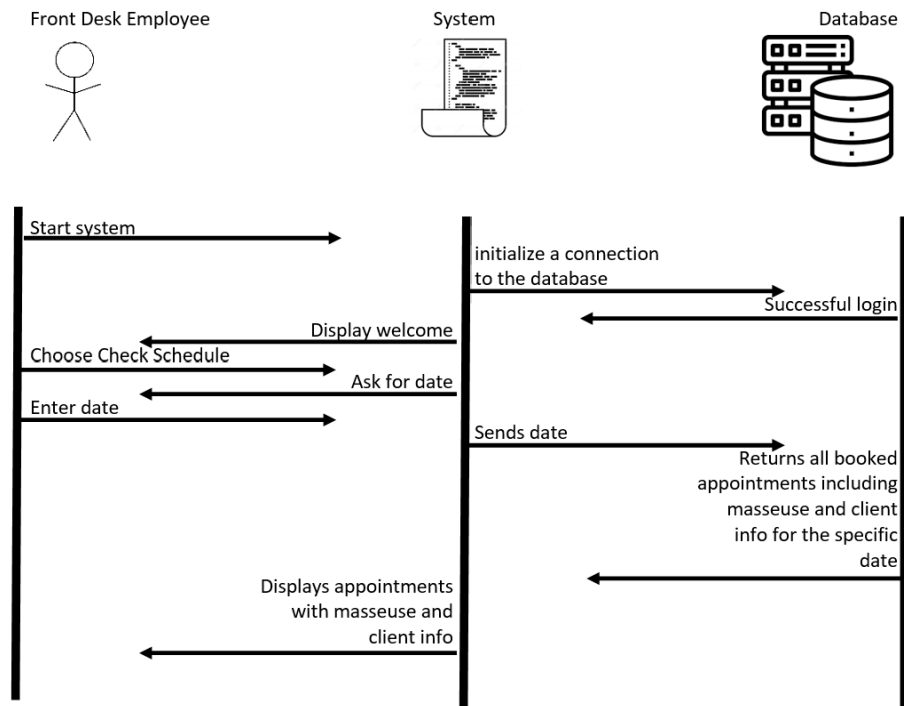


# System Sequence Diagrams

## UC-03: Add Masseuse

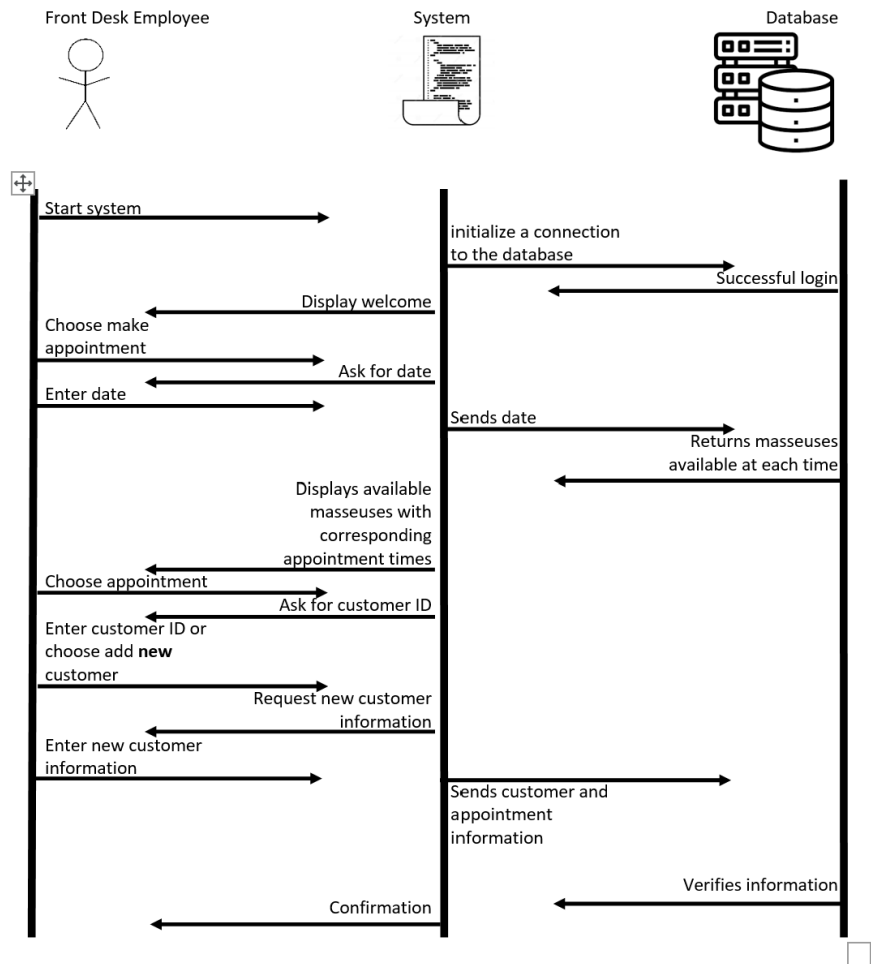


## UC-06: Check Schedule





## UC-08: Make Appointment



## Traceability Matrix

	P W	UC -01	UC -02	UC -03	UC -04	UC -05	UC -06	UC -07	UC -08	UC -09	UC -10	UC -11	UC -12	UC -13	UC -14
REQ -01	5	X					X		X	X					
REQ -02	3		X		X										
REQ -03	4			X		X		X							
REQ -04	4	X	X	X	X	X	X	X	X	X					
REQ -05	3								X	X					
REQ -06	3						X	X							
REQ -07	1						X	X			X	X			
REQ -08	2								X		X	X			
REQ -09	1	X													X
REQ -10	1	X									X	X			X
REQ -11	3								X						
REQ -12	2								X					X	
REQ -13	1												X		
REQ -14	2	X	X	X	X	X	X	X	X	X	X	X		X	X
REQ -15	1	X	X	X	X	X	X	X	X	X	X	X		X	X
REQ -16	4	X	X	X	X	X	X	X	X	X	X	X		X	X
REQ -17	2	X	X	X	X	X	X	X	X	X	X	X		X	X
Total PW		20	16	17	16	17	22	21	28	21	13	13	1	11	11

# Effort Estimation Using Use Case Points

## Project Size Estimation

### TCF

Technical Factor	Description	Weight	Complexity Weight
T1	Distributed system (running on multiple machines)	0	1
T2	Performance objectives (are response time and throughput performance critical?)	1	1
T3	End-user efficiency	2	3
T4	Complex internal processing	2	2
T5	Reusable design or code	1	3
T6	Easy to install (are automated conversion and installation included in the system?)	1	1
T7	Easy to use (including operations such as backup, startup, and recovery)	3	1
T8	Portable	1	3
T9	Easy to change (to add new features or modify existing ones)	3	3
T10	Concurrent use (by multiple users)	1	2
T11	Special security features	1	3
T12	Provides direct access for third parties (the system will be used from multiple sites in different organizations)	0	1
T13	Special user training facilities are required	0	1

$$TCF = .6 + (.01 * TCF)$$

$$\text{Total TCF} = 35$$

$$TCF = .95$$

## UCP

Use Case	Name	Actor Weight	Complexity Weight	UUCP
UC-01	Login	3	1	3
UC-02	AddClient	3	1	3
UC-03	AddMasseuse	3	1	3
UC-04	EditClient	3	2	6
UC-05	EditMasseuse	3	2	6
UC-06	CheckSchedule	3	3	9
UC-07	GetMasseuseSched	3	3	9
UC-08	MakeAppt	3	3	9
UC-09	RemoveAppt	3	2	6
UC-10	SetandGetShopHours	3	1	3
UC-11	SetandGetTimeslotLen	3	1	3
UC-12	EmailReminder	1	2	2
UC-13	SetandGetServices	3	1	3
UC-14	Admin	3	3	9

UUCP Total = 74

UCP = UUCP \* TCF \* ECF

UCP = 74 \* .95 \* 1 = 70.3

Many of these weights and/or complexities are generalized and guessed at as experience doing work like this is limited. However, thought was given to logical weights and complexity, and we believe that the UCP is a reasonably fair representation of what the actual effort would be. There may be discrepancies on specific use cases, but the average should be an accurate overall representation. Without Landon's experience in UI development the UCP would be much higher, but his expertise has allowed for a much more manageable development process.

# Domain Analysis

## Concept Definitions

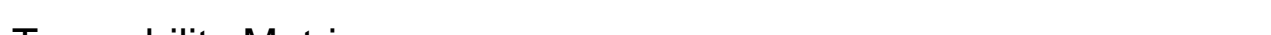
1. Add Customer – To add a customer via the client application and save it to the database to be selected upon making appointment.
2. Add Masseuse – To add a masseuse via the client application and save it to the database.
3. Add Services – Add services to the database that are selected when making an appointment.
4. Check Schedule – Pull up a list of the current day, or selected days, available to be scheduled.
5. Database Connection – To both establish and maintain connectivity with the SQL database.
6. Delete Appointment – To remove an appointment from the schedule and free up the space made available for another possible appointment schedule.
7. Login – To verify whether the user can access the program and additionally the type of permissions that the user can possess.
8. Make Appointment – To schedule an appointment via the client application and save it to the database avoiding overbooking.
9. Add user – To add a staff member to the database who can manage customers, masseuses, and other staff members accordingly.

## Association Definitions

Concept Pair	Association Description
Login ↔ 'Everything Else'	System requests login to verify access to system resources.
Database Connection ↔ 'Everything Else'	System must verify and connect with database before any information can be created, updated, or retrieved.
Make Appointment ↔ Check Schedule	Check for masseuse availability at a given time to ensure no overbooking.
Delete Appointment ↔ Check Schedule	Ensure schedule resources are allocated correctly for available/booked.
Add Customer ↔ Make Appointment	Ensure when appointment is made customer is or will immediately be input into the database.

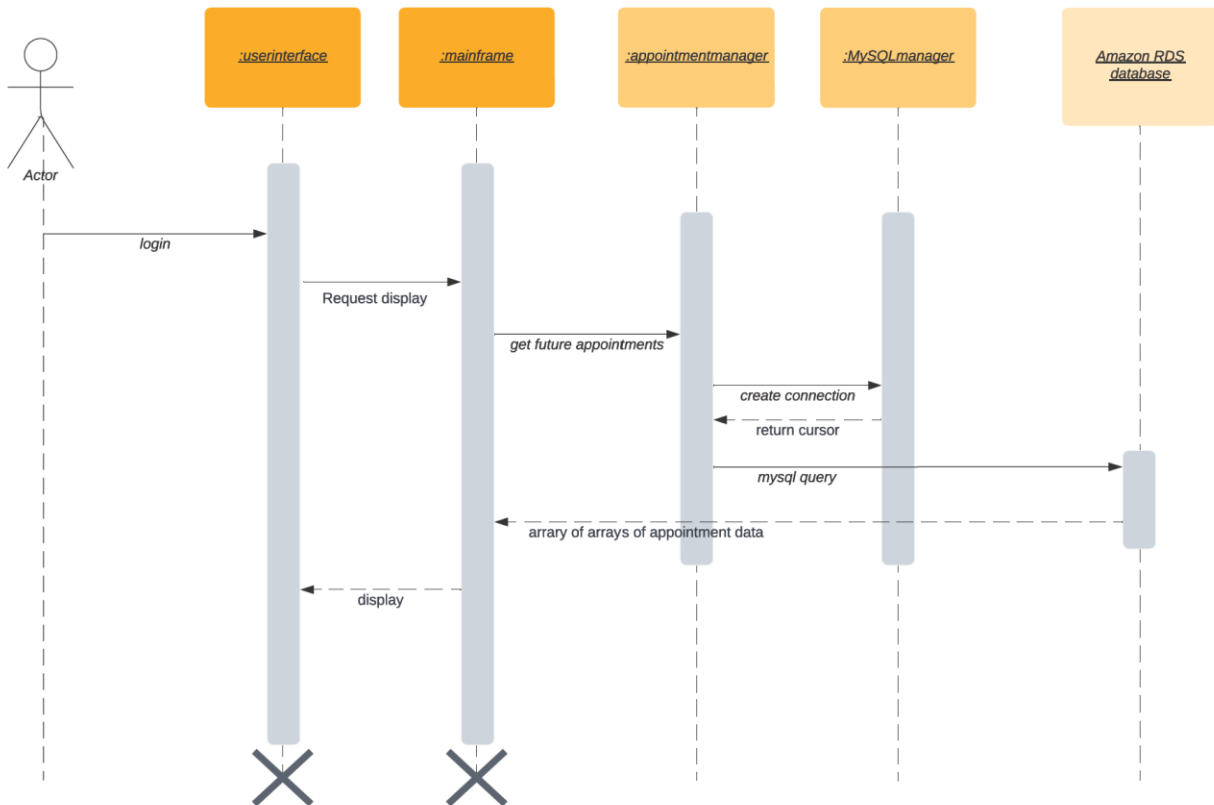
## Attribute Definitions

- address – used for customers and masseuses in database.
- appointmentId – integer id of appointment.
- customerId- integer id of customer.
- dbIsVerified – Verification of the connection to database.
- dow – integer indicating days of week masseuse available.
- email – used for customers and masseuses in database.
- end – last possible appointment that can be scheduled on a given day.
- isConnected – has active internet connection.
- isVerified – Verified user or admin for the system.
- masseuseId – integer id of masseuse.
- name – used for customers and masseuses in database.
- room – room used for appointment.
- start – when the masseuse can be scheduled their first appointment of the day.
- start\_time – start of appointment.
- search parameters – used when searching for masseuse schedule or daily schedule.
- status – booked or not.

[illegible]

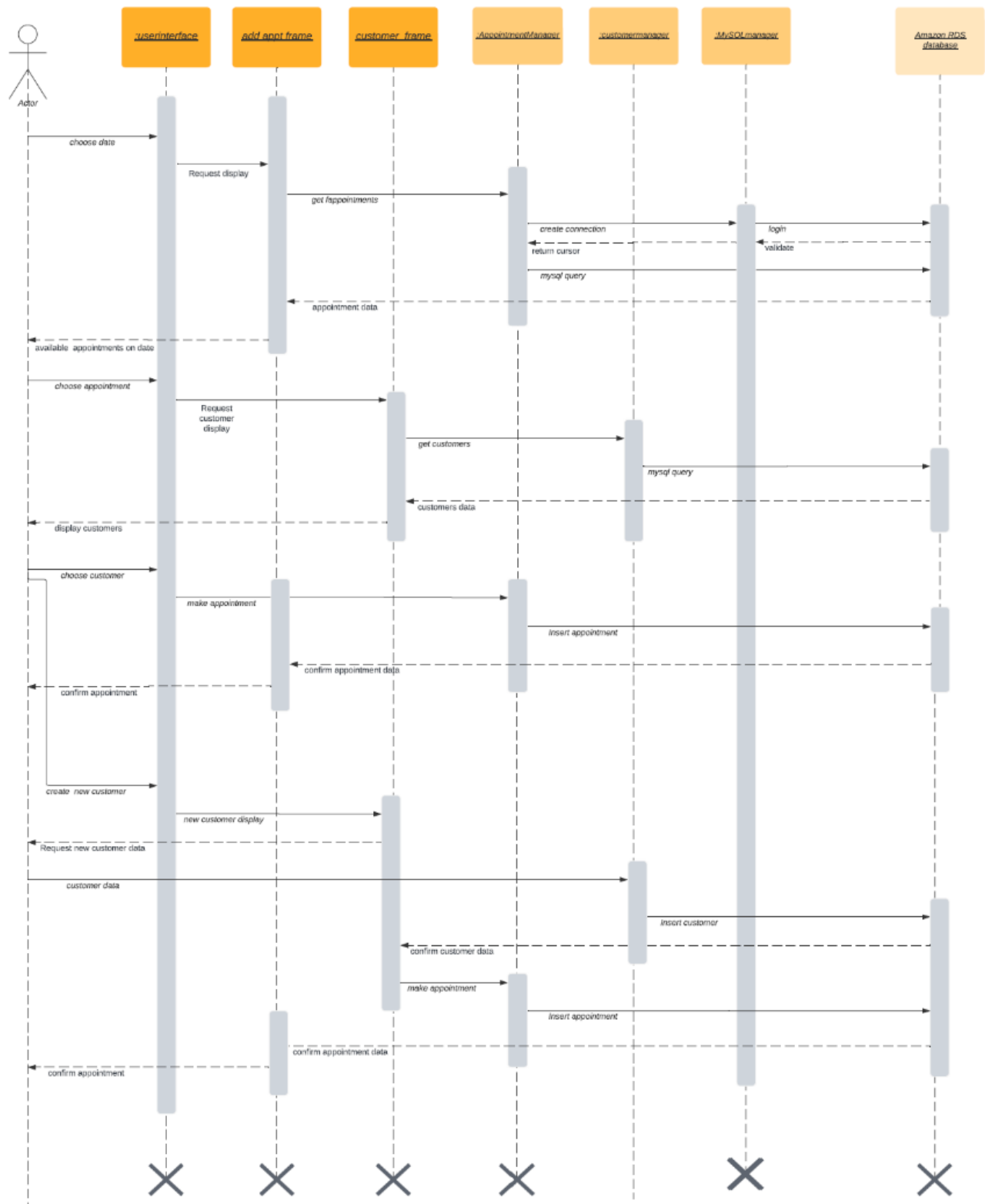
# Interaction Diagrams

## UC – 06 Show Schedule





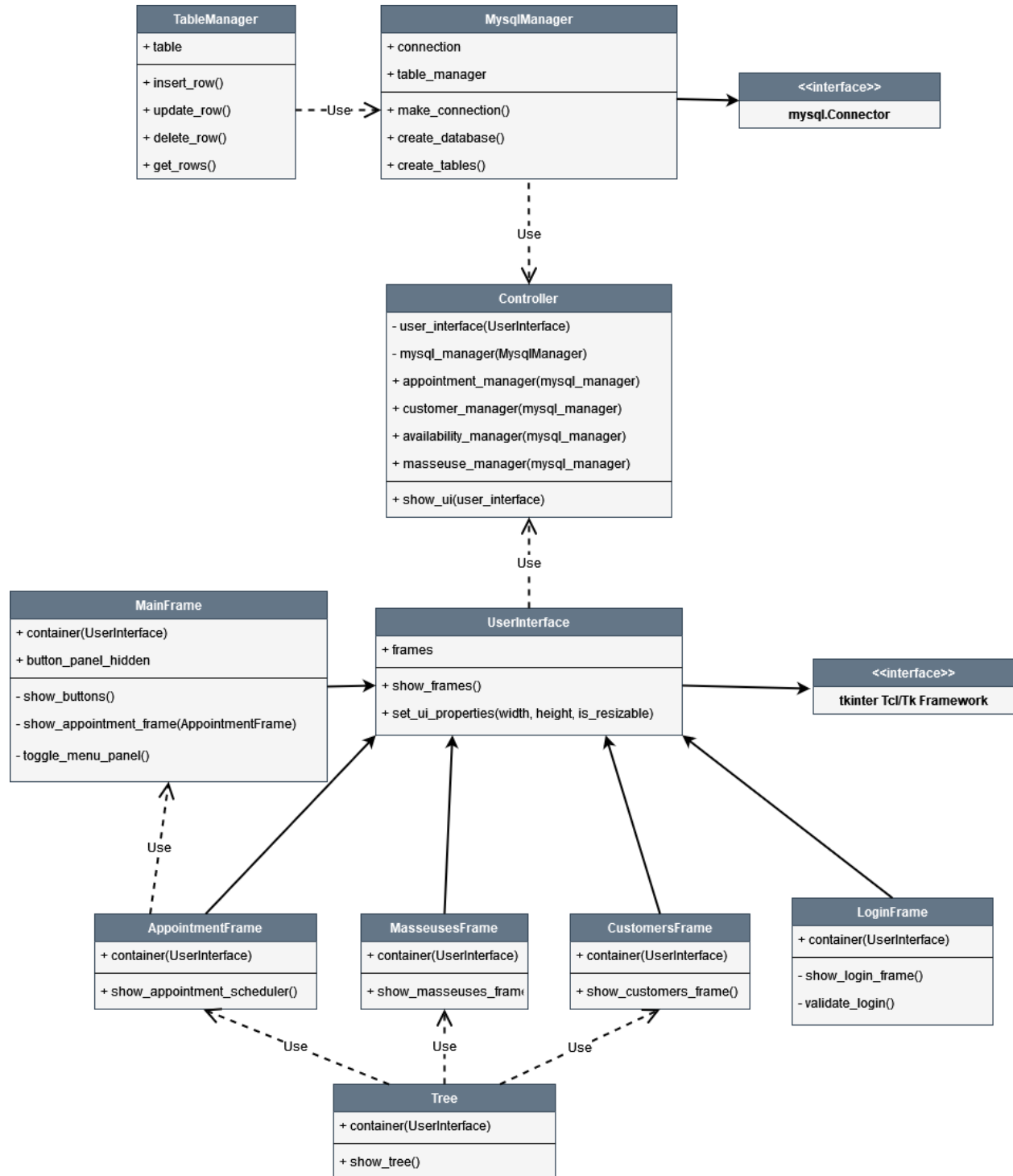
## UC – 08 Add Appointment



# Class Diagram and Interface Specification

## Class Diagram

[OneDrive link to manipulate the class diagram](#)



## Data Types and Operation Signatures

### Controller

Controller
- user_interface(UserInterface)
- mysql_manager(MySqlManager)
+ appointment_manager(mysql_manager)
+ customer_manager(mysql_manager)
+ availability_manager(mysql_manager)
+ masseuse_manager(mysql_manager)
+ show_ui(user_interface): bool

#### Definition

A class that acts as a connector and controller for communication between the `UserInterface` and the `MySQL` manager.

#### Associated Concepts

`MySQLManager`, `TableManager`, `UserInterface`, `MainFrame`, `AppointmentFrame`, `MasseusesFrame`, `CustomersFrame`, `LoginFrame`, `Tree`

#### Attributes

- user\_interface(ui: `UserInterface`)
- mysql\_manager(manager: `MySQLManager`)
- + appointment\_manager(this.mysql\_manager: `MySQLManager`)
- + customer\_manager(this.mysql\_manager: `MySQLManager`)
- + availability\_manager(this.mysql\_manager: `MySQLManager`)
- + masseuse\_manager(this.mysql\_manager: `MySQLManager`)

#### Operation Signatures

- + show\_ui(this.user\_interface: `UserInterface`)

Definition: Show the user interface using attribute `UserInterface`

### UserInterface

<div></div> <div>UserInterface</div>
+ frames<List<Frame>>>
+ show_frames(Frame): void
+ set_ui_properties(width: int, height: int, is_resizable: bool)

### Definition

A dependency of Controller that contains and displays the frames (windows) on the end-users screen.

### Associated Concepts

MainFrame, AppointmentFrame, MasseusesFrame, CustomersFrame, LoginFrame, Tree

### Attributes

+ frames (list: Frame)

### Operation Signatures

+ show\_frames(frame: Frame)

Definition: Initialize frames to be displayed via the user interface.

+ set\_ui\_properties(width: int, height: int, is\_resizable: bool)

Definition: Set the default properties of the main user interface which can be used by classes that implement the UserInterface

## MainFrame

<div></div> <div>MainFrame</div>
+ container(UserInterface)
+ button_panel_hidden: bool
- show_buttons(): void
- show_appointment_frame(AppointmentFrame): void
- toggle_menu_panel(): void

### Definition

A class for the main frame of the UserInterface that depends on the UserInterface. It displays the appointment frame and the menu panel by default.

### Associated Concepts

AppointmentFrame, Tree, UserInterface

### Attributes

+ container(user\_interface: UserInterface)

+ button\_panel\_hidden: bool

### Operation Signatures

- show\_buttons(): void

Definition: Show the side panel buttons in the frame referenced

- show\_appointment\_frame (frame: AppointmentFrame): void

Definition: Show the appointment container/frame given the AppointmentFrame object

- toggle\_menu\_panel(): void

Definition: Show or hide the menu panel from the frame based on whether or not  
button\_panel\_hidden is true

## AppointmentFrame

AppointmentFrame
+ container(UserInterface)
+ show_appointment_scheduler(): void

### Definition

A class that defines how to display the appointment scheduler from within the UserInterface.

### Associated Concepts

MainFrame, Tree, UserInterface

### Attributes

+ container(user\_interface: UserInterface)

### Operation Signatures

+ show\_appointment\_scheduler(): void

Definition: Display the appointment scheduler within the UserInterface given configured properties.

## MasseusesFrame

MasseusesFrame
+ container(UserInterface)
+ show_masseuses_frame(): void

### Definition

A class that defines how to display the masseuses frame from within the UserInterface.

### Associated Concepts

Tree, UserInterface

### Attributes

+ container(user\_interface: UserInterface)

### Operation Signatures

+ show\_masseuses\_frame(): void

Definition: Display the masseuses frame within the UserInterface given configured properties.

## CustomersFrame

CustomersFrame
+ container(UserInterface)
+ show_customers_frame(): void

### Definition

A class that defines how to display the customers frame from within the UserInterface.

### Associated Concepts

Tree, UserInterface

### Attributes

+ container(user\_interface: UserInterface)

### Operation Signatures

+ show\_customers\_frame(): void

Definition: Display the customers frame within the UserInterface given configured properties.

## LoginFrame

LoginFrame
+ container(UserInterface)
- show_login_frame(): void
- validate_login(): bool

### Definition

A class that defines how to display the login frame and window within the UserInterface.

### Associated Concepts

UserInterface

### Attributes

+ container(user\_interface: UserInterface)

### Operation Signatures

- show\_login\_frame(): void

Definition: Display the login frame within the UserInterface given configured properties.

- validate\_login(): bool

Definition: Returns true if provided login credentials within input boxes are correctly authenticated, else return false

## MysqlManager

MysqlManager
+ connection: mysql.Connector
+ table_manager: TableManager
+ make_connection(): void
+ create_database(): void
+ create_tables(): void

### Definition

A class that allows connection between the Mysql database. It also contains helper functions to assist with overall operations.

### Associated Concepts

TableManager, Controller

### Attributes

+ connection: mysql.Connector

+ table\_manager(manager: TableManager)

### Operation Signatures

+ make\_connection(): void

Definition: Establishes a connection with the mysql database and sets the connection attribute to the mysql.Connector object that was initialized.

+ create\_database(): void

Definition: Create the database given predefined database specifications.

+ create\_tables(): void

Definition: Create the tables within the database given predefined table specifications.

# TableManager

TableManager
+ table
+ insert_row(data: list): void
+ update_row(data: list): void
+ delete_row(index: int) void
+ get_rows(table_name: str): list

## Definition

A class that allows the manipulation of a table given table\_name on initialization

## Associated Concepts

MysqlManager

## Attributes

+ table: string

## Operation Signatures

+ insert\_row(data: list): void

Definition: A helper function to insert a row to the table given a list of data.

+ update\_row(data: list): void

Definition: A helper function to update a row in the table given a list of data.

+ delete\_row(index: int): void

Definition: A helper function to delete a row by number given an index integer.

+ get\_rows(table\_name: string): list

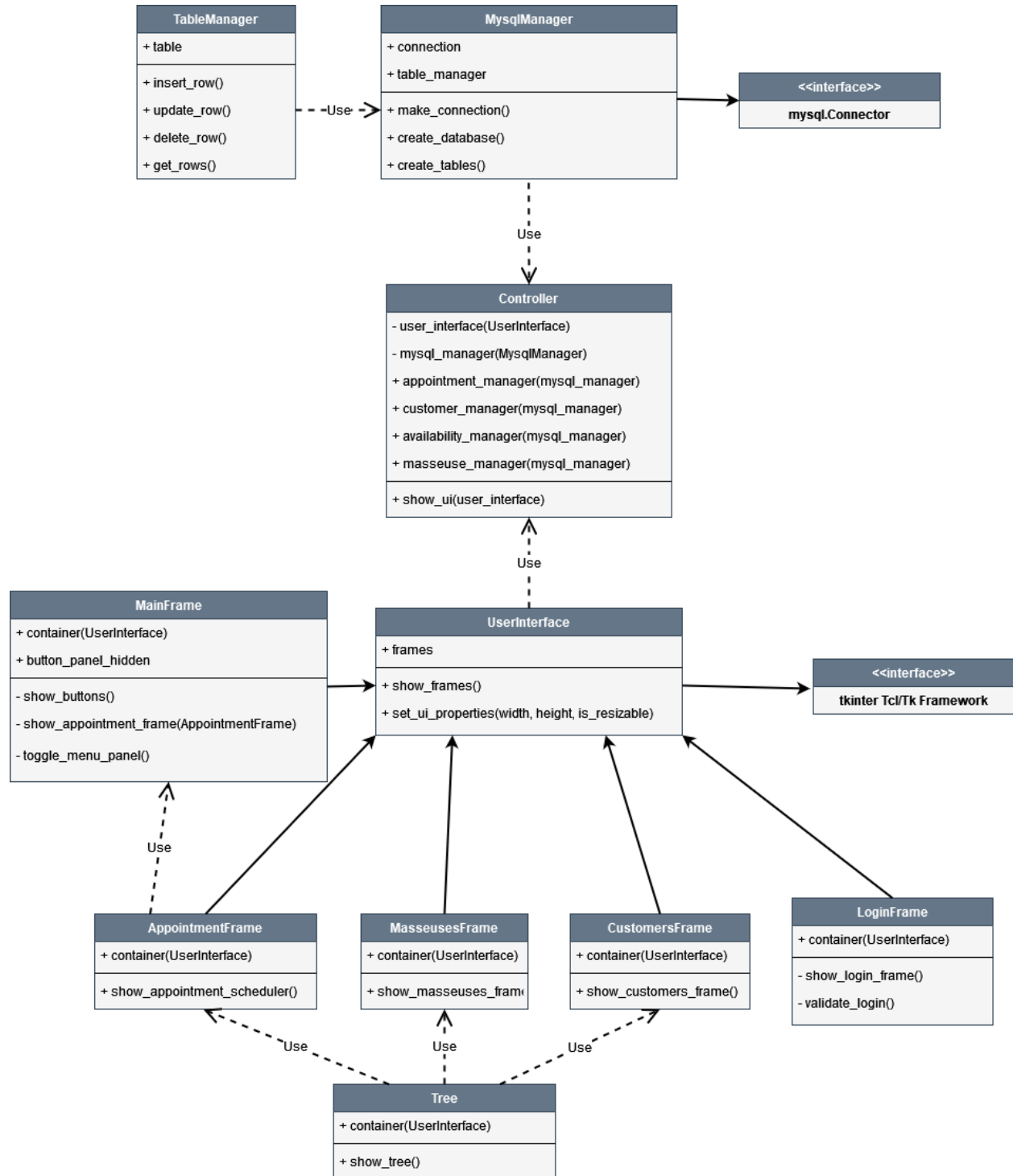
Definition: A helper function to return the rows of a table as a list given table\_name string.



# System Architecture and System Design

## Class Diagram

[OneDrive link to manipulate the class diagram](#)



# Data Types and Operation Signatures

## Controller

Controller
- user_interface(UserInterface)
- mysql_manager(MysqlManager)
+ appointment_manager(mysql_manager)
+ customer_manager(mysql_manager)
+ availability_manager(mysql_manager)
+ masseuse_manager(mysql_manager)
+ show_ui(user_interface): bool

### Definition

A class that acts as a connector and controller for communication between the UserInterface and the MySql manager.

### Associated Concepts

MysqlManager, TableManager, UserInterface, MainFrame, AppointmentFrame, MasseusesFrame, CustomersFrame, LoginFrame, Tree

### Attributes

- user\_interface(ui: UserInterface)
- mysql\_manager(manager: MysqlManager)
- + appointment\_manager(this.mysql\_manager: MysqlManager)
- + customer\_manager(this.mysql\_manager: MysqlManager)
- + availability\_manager(this.mysql\_manager: MysqlManager)
- + masseuse\_manager(this.mysql\_manager: MysqlManager)

### Operation Signatures

- + show\_ui(this.user\_interface: UserInterface)

Definition: Show the user interface using attribute UserInterface

## UserInterface

<div></div> <div>UserInterface</div>
+ frames<List<Frame>>>
+ show_frames(Frame): void
+ set_ui_properties(width: int, height: int, is_resizable: bool)

### Definition

A dependency of Controller that contains and displays the frames (windows) on the end-users screen.

### Associated Concepts

MainFrame, AppointmentFrame, MasseusesFrame, CustomersFrame, LoginFrame, Tree

### Attributes

+ frames (list: Frame)

### Operation Signatures

+ show\_frames(frame: Frame)

Definition: Initialize frames to be displayed via the user interface.

+ set\_ui\_properties(width: int, height: int, is\_resizable: bool)

Definition: Set the default properties of the main user interface which can be used by classes that implement the UserInterface

## MainFrame

<div></div> <div>MainFrame</div>
+ container(UserInterface)
+ button_panel_hidden: bool
- show_buttons(): void
- show_appointment_frame(AppointmentFrame): void
- toggle_menu_panel(): void

### Definition

A class for the main frame of the UserInterface that depends on the UserInterface. It displays the appointment frame and the menu panel by default.

### Associated Concepts

AppointmentFrame, Tree, UserInterface

### Attributes

+ container(user\_interface: UserInterface)

+ button\_panel\_hidden: bool

### Operation Signatures

- show\_buttons(): void

Definition: Show the side panel buttons in the frame referenced

- show\_appointment\_frame (frame: AppointmentFrame): void

Definition: Show the appointment container/frame given the AppointmentFrame object

- toggle\_menu\_panel(): void

Definition: Show or hide the menu panel from the frame based on whether or not  
button\_panel\_hidden is true

## AppointmentFrame

AppointmentFrame
+ container(UserInterface)
+ show_appointment_scheduler(): void

### Definition

A class that defines how to display the appointment scheduler from within the UserInterface.

### Associated Concepts

MainFrame, Tree, UserInterface

### Attributes

+ container(user\_interface: UserInterface)

### Operation Signatures

+ show\_appointment\_scheduler(): void

Definition: Display the appointment scheduler within the UserInterface given configured properties.

## MasseusesFrame

MasseusesFrame
+ container(UserInterface)
+ show_masseuses_frame(): void

### Definition

A class that defines how to display the masseuses frame from within the UserInterface.

### Associated Concepts

Tree, UserInterface

### Attributes

+ container(user\_interface: UserInterface)

### Operation Signatures

+ show\_masseuses\_frame(): void

Definition: Display the masseuses frame within the UserInterface given configured properties.

## CustomersFrame

CustomersFrame
+ container(UserInterface)
+ show_customers_frame(): void

### Definition

A class that defines how to display the customers frame from within the UserInterface.

### Associated Concepts

Tree, UserInterface

### Attributes

+ container(user\_interface: UserInterface)

### Operation Signatures

+ show\_customers\_frame(): void

Definition: Display the customers frame within the UserInterface given configured properties.

## LoginFrame

LoginFrame
+ container(UserInterface)
- show_login_frame(): void
- validate_login(): bool

### Definition

A class that defines how to display the login frame and window within the UserInterface.

### Associated Concepts

UserInterface

### Attributes

+ container(user\_interface: UserInterface)

### Operation Signatures

- show\_login\_frame(): void

Definition: Display the login frame within the UserInterface given configured properties.

- validate\_login(): bool

Definition: Returns true if provided login credentials within input boxes are correctly authenticated, else return false

## MysqlManager

MysqlManager
+ connection: mysql.Connector
+ table_manager: TableManager
+ make_connection(): void
+ create_database(): void
+ create_tables(): void

### Definition

A class that allows connection between the Mysql database. It also contains helper functions to assist with overall operations.

### Associated Concepts

TableManager, Controller

### Attributes

+ connection: mysql.Connector

+ table\_manager(manager: TableManager)

### Operation Signatures

+ make\_connection(): void

Definition: Establishes a connection with the mysql database and sets the connection attribute to the mysql.Connector object that was initialized.

+ create\_database(): void

Definition: Create the database given predefined database specifications.

+ create\_tables(): void

Definition: Create the tables within the database given predefined table specifications.

## TableManager

TableManager
+ table
+ insert_row(data: list): void
+ update_row(data: list): void
+ delete_row(index: int) void
+ get_rows(table_name: str): list

### Definition

A class that allows the manipulation of a table given table\_name on initialization

### Associated Concepts

MySQLManager

### Attributes

+ table: string

### Operation Signatures

+ insert\_row(data: list): void

Definition: A helper function to insert a row to the table given a list of data.

+ update\_row(data: list): void

Definition: A helper function to update a row in the table given a list of data.

+ delete\_row(index: int): void

Definition: A helper function to delete a row by number given an index integer.

+ get\_rows(table\_name: string): list

Definition: A helper function to return the rows of a table as a list given table\_name string.

## Class to Domain Traceability Matrix

	CD-01	CD-02	CD-03	CD-04	CD-05	CD-06	CD-07	CD-08	CD-09
TableManager	X	X	X		X			X	
MysqlManager	X	X	X		X			X	
Controller	X	X	X	X	X	X	X	X	X
UserInterface	X	X	X	X		X	X	X	X
AppointmentFrame			X	X	X	X		X	
MasseusesFrame		X			X				
CustomersFrame	X				X				
LoginFrame							X		X
Tree	X	X	X	X	X			X	
MainFrame				X	X			X	

- Names were changed to better reflect coding strategy.
- Login and Add User were combined into the class LoginFrame to allocate all access from there.
- Check Schedule, Make Appointment, and Delete Appointment were correlated to each other using the Tree, MainFrame, and AppointmentFrame.
- Database Connection is class MysqlManager.
- Controller class was created to be the hub and connection for and between the application, functions, and the SQL server.
- Add Services was combined with Make Appointment in the class AppointmentFrame.



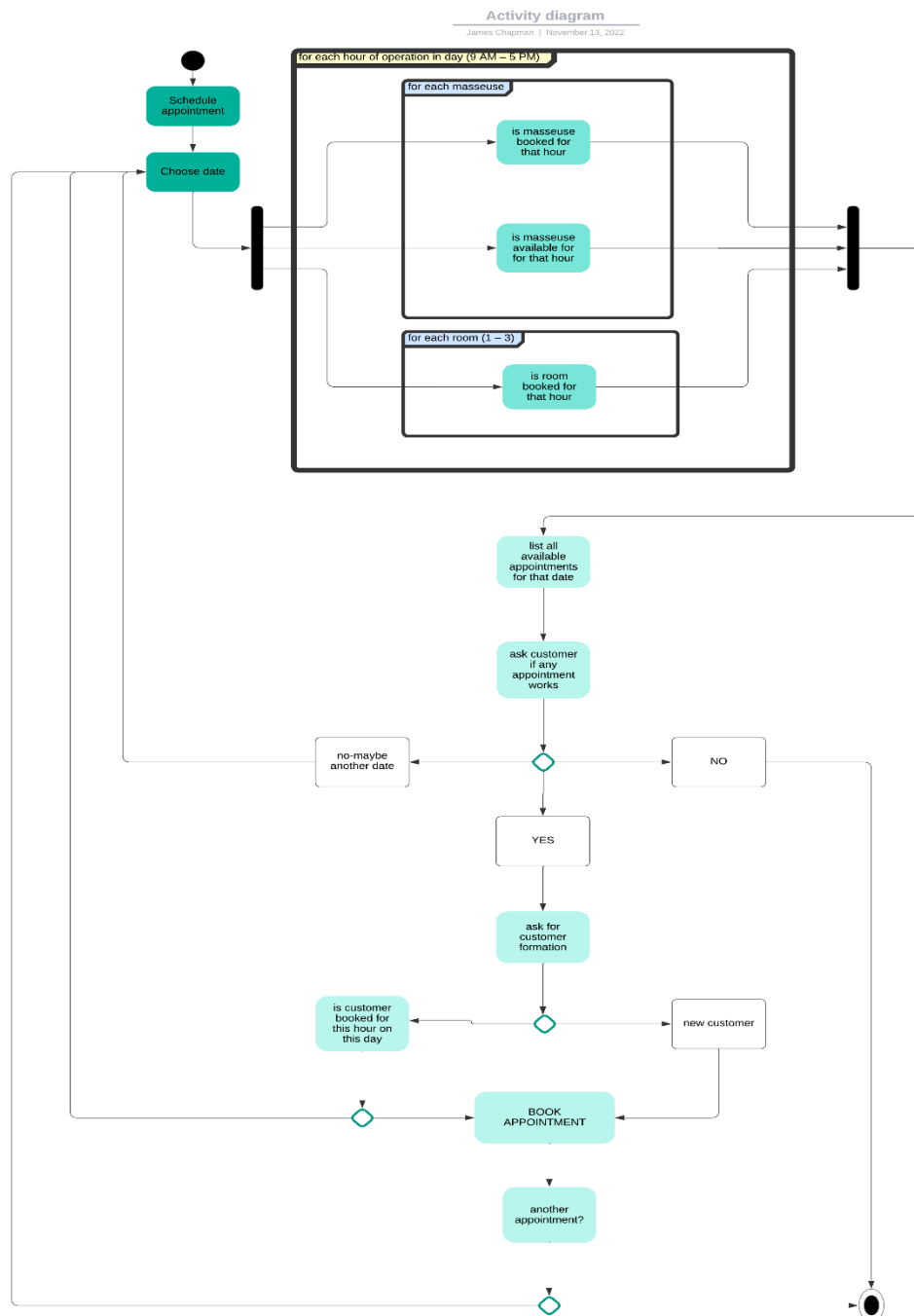
# Algorithms and Data Structures

## Algorithms

Most of the logic of the program is implemented by joining tables in MySQL. Using query language, we can collect relational data efficiently. This was a major reason why we chose a database to store customer information, masseuse information, and most importantly appointment schedules. Data produced by the database is in the form of arrays and there are simple algorithms to parse through the data in order to provide the interface with usable information.

To find the available appointments, every timeslot that has less than 3 appointments is appended to an array that provides a drop-down of the available appointments.

## Activity Diagram UC-08 MakeAppointment



# Data Structures

While the system does not incorporate a specific data structure known to Computer Science, it implements a table/relational SQL database to handle the storage of data.

Relational databases allow multiple tables to be joined with one another when querying and storing data. For instance, a customer is scheduled an appointment – by using a relational database, the specific appointment can be created, and the customer can be linked with the appointment so that the customer does not need to be written down statically in the appointment list, they only need to be referenced by a key so that the query is able to determine which customer is displayed with the appointment.

If a simple array were used, in order to display appointments scheduled, then all appropriate information would need to be set when the elements are assigned to the array indexes. While this method would work, it wouldn't be efficient and consume additional memory than it really needs to, thus giving justification to our decision to implement a database to accommodate the storage and manipulation of data.

Finally, using a SQL database allows us to easily store the database in the cloud using AWS. AWS allows for database the easily and securely be manipulated while reducing concerns that the database may one day be lost given the database is stored in multiple copies across multiple redundant and reliable servers.

During runtime, several strings and variable-based widgets are displayed to the end user. The data that is retrieved from the static storage location (the AWS instance) is then stored as an allocation in the memory store of the Python interpreter. If the memory is manipulated, it can then be written back to the database for other users to see as the datastore changes.

A dictionary was used to keep track of the number of appointments per timeslot. This is important so that appointments are not overbooked.

# User Interface: Design and Implementation

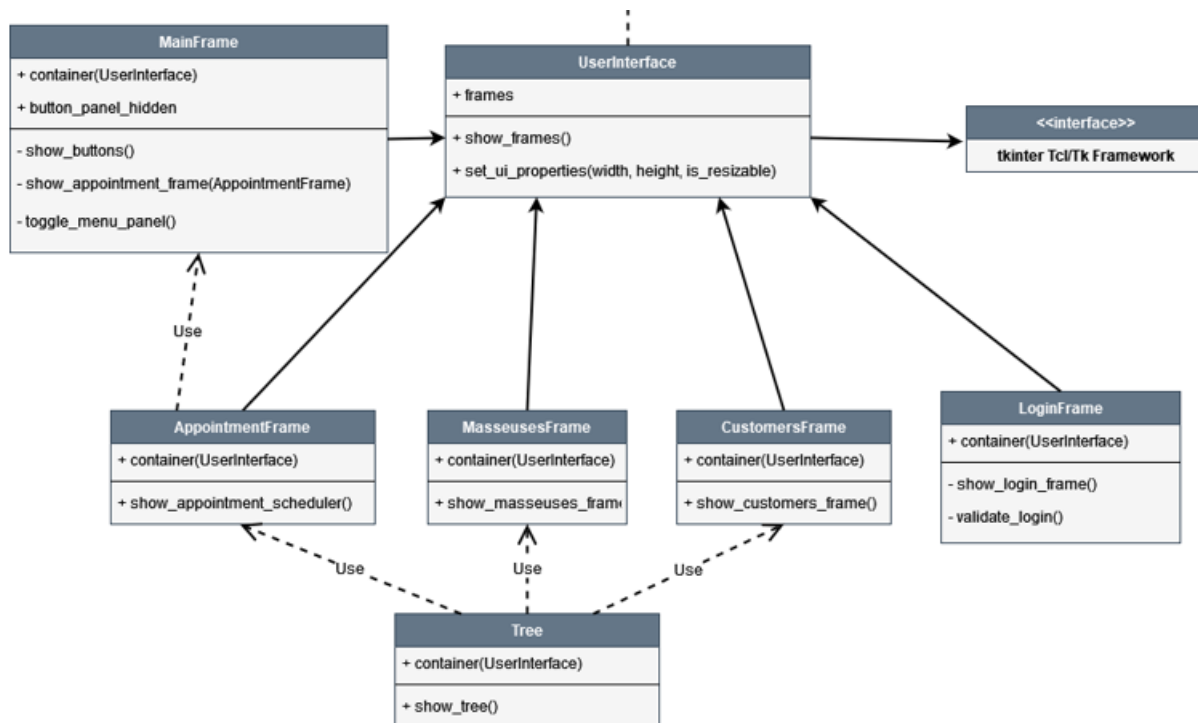
The goal of a user interface should be to allow an end user to utilize the usefulness of a program in a way that is as simple as possible to understand. It's appearance should display information in such a way that the user can view the screen and have few questions about the information presented, besides specific use cases for the software that require a deeper understanding about why it should be used.

Tkinter is a widely-used Python binding which interfaces with the Tk toolkit – the toolkit is written in C, is largely cross-platform, and is used to keep the native feel of the host operating system while also allowing easy-to-use implementation methodologies to the programmer. Because the project is written in Python, Tkinter provides the API to talk with the lower level toolkit and as such is extremely versatile to understand and use.

Given timing requirements and expectations of the software and documentation associated, Tkinter was the best way to maintain cross compatibility while still allowing for uniformity between the different operating systems supported: Microsoft Windows, macOS, and Linux along with flavors of Linux implementing an OS graphical user interface.

## User Interface Class Structure

The following diagram, as also shown under the Class Diagram and Interface Specification section, demonstrates the intended class relationship of the user interface:



As shown, the UserInterface contains a direct implementation of the Tkinter Python module and extends the capability of the core module of Tk, as such it is considered the parent driver class of the user interface.

However, the UserInterface class, as is the case with Tk, cannot run by itself. It requires an object known as a Frame to display at runtime, or else the window will not appear on screen to the user. As a result, several child frames are declared, and their frame contents specified, so that the UserInterface frame container can display the needed frames at the demand of the current state of the UI.

The current state of the UI as referenced in the paragraph above is determined based on the input of the user. If the end user wants to navigate to a different window, they will click on a variety of buttons which will trigger an event listener and cause the state of the user interface to change according to the action defined when the buttons are pressed.

# User Interface Layout

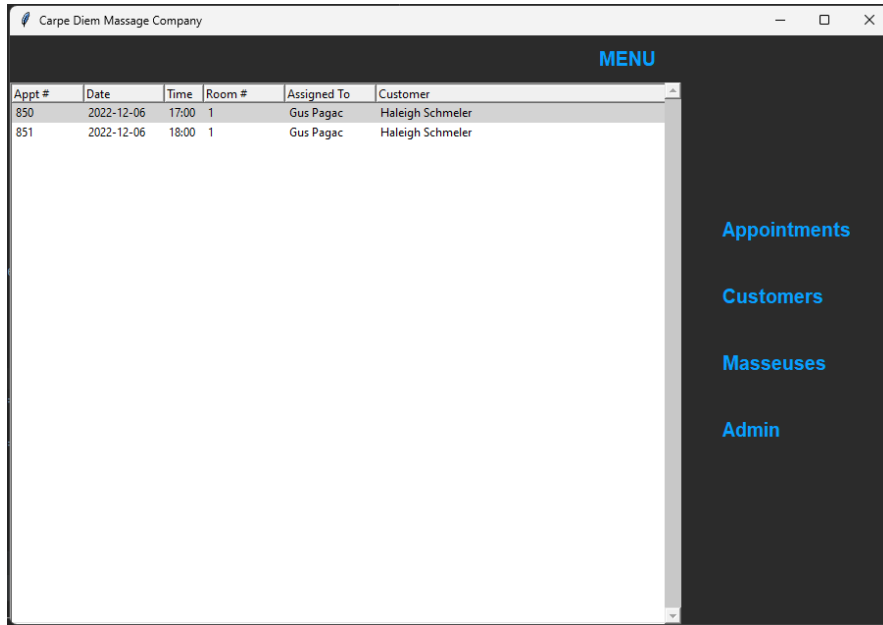
## LoginFrame:



The screenshot shows a window titled "Carpe Diem Massage Company" with standard window controls (minimize, maximize, close). Inside the window, there are two text input fields. The first field is labeled "Enter Username" and the second is labeled "Enter Password". Below these fields is a button labeled "Login".

- The LoginFrame is a standalone frame that accepts a username and password as input.
- Login Button:
  - The login button calls a function that queries the current contents of the username and password inputs.
  - The input strings are sent to an authenticator function that checks the strings against an authentication table.
  - If the login information does not match, the login prompt displays an error.
  - If the login was successful, the login frame closes and the appointment scheduler frame is allowed to display.

## Appointment/Masseuses/Customers Frame:



Carpe Diem Massage Company

MENU

Appt #	Date	Time	Room #	Assigned To	Customer
850	2022-12-06	17:00	1	Gus Pagac	Haleigh Schmeler
851	2022-12-06	18:00	1	Gus Pagac	Haleigh Schmeler

Appointments

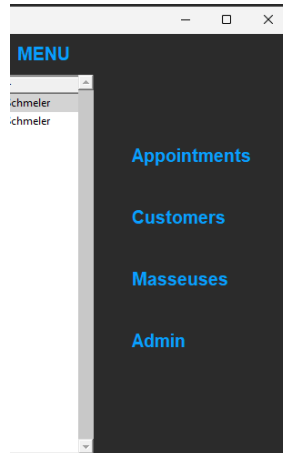
Customers

Masseuses

Admin

- The above is the layout of the Appointments/Customers/Masseuses Frames.
- Each frame displays a custom Tree (reference to the Tree class outlined earlier), and each Tree defines how and in what way to display a table specific to each type of Person.
  - Each tree displays columns specific to the type of individual frame to be displayed.
  - Along with specific columns by window, the data queried to display the rows is of course different as well – where the AppointmentFrame utilizes the appointment table, etc.

- Menu Button:
  - The menu button is used to expand or collapse a navigation bar vertical with the UI.
  - The end user can pick a nav item to navigate to the appropriate frame, as needed.

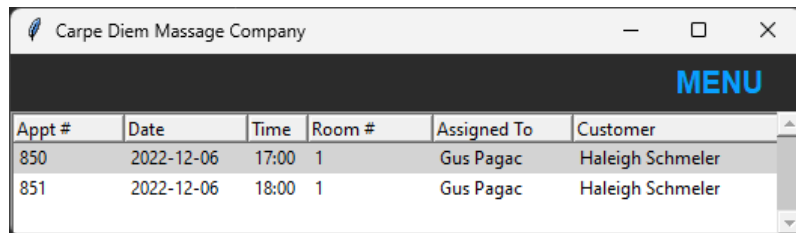


- The following is an example of the full Frame, where the sidebar is collapsed.

A screenshot of a web application window titled "Carpe Diem Massage Company". The window displays a table with appointment data. The table has columns for Appt #, Date, Time, Room #, Assigned To, and Customer. The data rows show appointments for 2022-12-06. A "MENU" button is visible in the top right corner of the content area.

Appt #	Date	Time	Room #	Assigned To	Customer
850	2022-12-06	17:00	1	Gus Pagac	Haleigh Schmeler
851	2022-12-06	18:00	1	Gus Pagac	Haleigh Schmeler

Notice, the user interface is completely resizable allowing for a low-profile design where the entire screen need not be utilized.



Appt #	Date	Time	Room #	Assigned To	Customer
850	2022-12-06	17:00	1	Gus Pagac	Haleigh Schmeler
851	2022-12-06	18:00	1	Gus Pagac	Haleigh Schmeler

## User Interface Reusability and Modularity

The layout and structure of the `UserInterface` component of this software is being designed in a way that it would allow for the ability to add on additional windows, frames, and other UI additions by inheriting classes and implementing their construction.

As shown in the User Interface Class Structure section, additional frames can be added that are custom to the individual frame but also inherit the same attributes/properties of the `UserInterface` main driver itself.



# Design of Tests

All functions will be tested for valid input and valid output responses by applying random inputs of varying data types to test for valid output data. This was largely completed during implementation in a test-as-we-go fashion.

If time allows there will be added testing based on the non-functional requirements to add functionality and user-friendliness including things such as updating store hours, sending reminder emails, and adding services available.

## Class Unit Tests

Test Case 1:	Login
Unit to Test:	LoginFrame, MainFrame, UserInterface, AppointmentFrame
Assumptions:	Power and internet are available, and the user has initiated the program.
Steps:	<ol style="list-style-type: none"><li>1. Execute Program.</li><li>2. Fill in authorization information on prompt.</li></ol>
Expected Result:	With successful login the program shall move forward to the MainFrame/AppointmentFrame.
Fail Result:	User informed of failed login attempt and prompted to re-input credentials.

Test Case 2:	Add Masseuse / Add Client
Unit to Test:	UserInterface, MainFrame, TableManager, MySqlManager, Controller, MasseusesFrame, CustomersFrame, Tree
Assumptions:	Successfully logged on with database access, and the AppointmentFrame and MainFrame are displayed.
Steps:	<ol style="list-style-type: none"><li>1. Press Customers or Masseuses.</li><li>2. Press Add at the top of the MainFrame.</li><li>3. Fill out information.</li><li>4. Select Done to push to the database.</li></ol>
Expected Result:	Database updated successfully displaying the MainFrame and either MasseuseFrame or CustomersFrame respectively with updated information.
Fail Result:	Prompt from database with any database errors and/or database is not updated.

Test Case 3:	Delete Masseuse / Delete Client
Unit to Test:	UserInterface, MainFrame, TableManager, MySqlManager, Controller, MasseusesFrame, CustomersFrame, Tree
Assumptions:	Successfully logged on with database access, and the AppointmentFrame and MainFrame are displayed.
Steps:	<ol style="list-style-type: none"><li>1. Press Customers or Masseuses.</li><li>2. Select the correct entry from the CustomersFrame or MasseusesFrame.</li></ol>

	<ol style="list-style-type: none"> <li>Press Delete on the MainFrame.</li> <li>Respond Yes to "Are you sure?" prompt pushing the update do the database.</li> </ol>
Expected Result:	Database updated successfully displaying the MainFrame and either MasseuseFrame of CustomersFrame respectively with updated information.
Fail Result:	Prompt from database with any database errors and/or database is not updated.

Test Case 4:	Modify Masseuse / Modify Customer
Unit to Test:	UserInterface, MainFrame, TableManager, MySqlManager, Controller, MasseusesFrame, CustomersFrame, Tree
Assumptions:	Successfully logged on with database access, and the AppointmentFrame and MainFrame are displayed.
Steps:	<ol style="list-style-type: none"> <li>Press Customers or Masseuses.</li> <li>Select the correct entry from the CustomersFrame or MasseusesFrame.</li> <li>Press Delete on the MainFrame.</li> <li>Respond Yes to "Are you sure?" prompt pushing the update do the database.</li> </ol>
Expected Result:	Database updated successfully displaying the MainFrame and either MasseuseFrame of CustomersFrame respectively with updated information.
Fail Result:	Prompt from database with any database errors and/or database is not updated.

Test Case 5:	Add Appointment
Unit to Test:	UserInterface, MainFrame, TableManager, MySqlManager, Controller, AppointmentFrame, Tree
Assumptions:	Successfully logged on with database access, and the AppointmentFrame and MainFrame are displayed.
Steps:	<ol style="list-style-type: none"> <li>Press Add.</li> <li>Input appointment information.</li> <li>Select Done to push to the database.</li> </ol>
Expected Result:	Database updated successfully displaying the MainFrame and AppointmentFrame with updated information.
Fail Result:	Prompt from database with any database errors and/or database is not updated correctly.

Test Case 1:	Delete Appointment
Unit to Test:	UserInterface, MainFrame, TableManager, MysqlManager, Controller, AppointmentFrame, Tree
Assumptions:	Successfully logged on with database access and AppointmentFrame and MainFrame displayed.
Steps:	<ol style="list-style-type: none"> <li>1. Select appointment to be deleted from the AppointmentFrame.</li> <li>2. Press Delete in the MainFrame.</li> <li>3. Respond Yes to "Are you sure?" prompt pushing the update to the database.</li> </ol>
Expected Result:	Database updated successfully displaying the MainFrame and AppointmentFrame with updated information.
Fail Result:	Prompt from database with any database errors and/or database is not updated correctly.

Test Case 6 may require deleting of the old appointment and creation of a new appointment.

Test Case 6:	Modify Appointment
Unit to Test:	UserInterface, MainFrame, TableManager, MysqlManager, Controller, AppointmentFrame, Tree
Assumptions:	Successfully logged on with database access and AppointmentFrame and MainFrame displayed.
Steps:	<ol style="list-style-type: none"> <li>1. Select appointment to be modified from the AppointmentFrame.</li> <li>2. Press Modify in the MainFrame.</li> <li>3. Update entry values as needed.</li> <li>4. Select Done to push update to database.</li> </ol>
Expected Result:	Database updated successfully displaying the MainFrame and AppointmentFrame with updated information.
Fail Result:	Prompt from database with any database errors and/or database is not updated correctly.

Test Case 7:	Add User
Unit to Test:	UserInterface, MainFrame, TableManager, MysqlManager, Controller, AdminFrame
Assumptions:	Successfully logged on with database access, and the AppointmentFrame and MainFrame are displayed.
Steps:	<ol style="list-style-type: none"> <li>1. Select Administration from MainFrame.</li> <li>2. Select Add from MainFrame.</li> <li>3. Fill in information.</li> <li>4. Select Done to push to database.</li> </ol>
Expected Result:	Database updated successfully displaying the MainFrame and AdminFrame with updated information.
Fail Result:	Prompt from database with any database errors and/or database is not updated correctly.

Test Case 8:	Delete User
--------------	-------------

Unit to Test:	UserInterface, MainFrame, TableManager, MySqlManager, Controller, AdminFrame
Assumptions:	Successfully logged on with database access, and the AppointmentFrame and MainFrame are displayed.
Steps:	<ol style="list-style-type: none"> <li>1. Select Administration from MainFrame.</li> <li>2. Select User to remove from AdminFrame.</li> <li>3. Select Delete from MainFrame.</li> </ol>
Expected Result:	Database updated successfully displaying the MainFrame and AdminFrame with updated information.
Fail Result:	Prompt from database with any database errors and/or database is not updated correctly.

Test Case 9 may require deleting User and creating a new User with the required permissions.

Test Case 9:	Modify User
Unit to Test:	UserInterface, MainFrame, TableManager, MySqlManager, Controller, AdminFrame
Assumptions:	Successfully logged on with database access, and the AppointmentFrame and MainFrame are displayed.
Steps:	<ol style="list-style-type: none"> <li>1. Select Administration.</li> <li>2. Select User to modify from the AdminFrame.</li> <li>3. Update entry values as needed.</li> <li>4. Select Done to push to database.</li> </ol>
Expected Result:	Database updated successfully displaying the MainFrame and AdminFrame with updated information.
Fail Result:	Prompt from database with any database errors and/or database is not updated correctly.

# History of Work

Major points only with some items grouped together if occurred on same date.

- Sept 4, 2022 – Completed Proposal
- Sept 10, 2022 – GIT setup with first commit
- Sept 11, 2022 – First UI commit
- Sept 21, 2022 – Database setup
- Sept 25, 2022 – First Report Submitted
- Oct 2, 2022 – Show Appointments frame implementation begins
- Oct 3, 2022 – Navigation buttons and login frame implemented
- Oct 4, 2022 – UI adjustments made
- Oct 6, 2022 – MySQL linked to interface
- Oct 16, 2022 – Second Report Submitted
- Oct 27, 2022 – Attempted Calendar implementation in Add Appointment UI.
- Oct 29, 2022 – Began initialized appointment management frame with right-click menu.
- Oct 30, 2022 – Finalized Calendar implementation and updated appointment frame.
- Nov 2, 2022 – More UI updates including tree updates.
- Nov 3, 2022 – UI Frames adjustment
- Nov 3, 2022 – First Demo Completed
- Nov 13, 2022 – Part One of Report Three Submitted
- Nov 20, 2022 – Report Three Final Submitted

## Current Status

Currently we are working on finalizing the implementation of Add Appointment to include masseuse and client information and the implementation of Delete appointment.

## Future Work

1. Implement no double-booking requirement by Nov 27, 2022.
2. Implement Create/Delete Customer end of Nov 30, 2022.
3. Implement Create/Delete Masseuse end of Dec 2, 2022.
4. Implement Admin/User accessibility before Final Demo.
5. Far Future Work (Possibly not completed before end of semester.)
  - a. Implement ability to change store hours and days open.
  - b. Create/Delete services offered.
  - c. Specify services requested in Create Appointment.
  - d. Email reminders to customers about upcoming appointments.

# References

- <https://github.com/>
- <https://www.jetbrains.com/pycharm/>
- <https://discord.com/>
- <https://www.ece.rutgers.edu/~marsic/Teaching/SE/syllabus.html>
- <https://aws.amazon.com/>
- <https://www.office.com/>