

# 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/DsF5PsT8>

### Submission:

16 October 2022 - Report Two - Final

# Table of Contents

Carpe Diem Massage Company .....	1
Fort Hays State University - Department of Computer Science .....	1
Software Engineering Fall 2022 .....	1
Distribution of Work .....	4
Analysis and Domain Modeling Conceptual Model.....	5
Concept Definitions .....	5
Association Definitions .....	5
Attribute Definitions .....	6
Use Case to Domain Traceability Matrix .....	6
Domain Diagram .....	7
System Operation Contracts.....	8
Data Model and Persistent Data Storage .....	9
Mathematical Model .....	9
Interaction Diagrams .....	10
UC – 06 Show Schedule .....	10
UC – 08 Add Appointment.....	11
Class Diagram and Interface Specification .....	12
Class Diagram.....	12
Data Types and Operation Signatures .....	13
Class to Domain Traceability Matrix .....	19
Algorithms.....	19
Data Structures .....	21
User Interface: Design and Implementation .....	21
User Interface Class Structure .....	22
User Interface Layout .....	23
User Interface Reusability and Modularity.....	26
Design of Tests .....	27
Class Unit Tests .....	27
Project Management .....	31
Merging the Contributions from Individual Team Members .....	31
Project Coordination Progress Report.....	31

Plan of Work .....32

Breakdown of Responsibilities.....32

# Distribution of Work

- James Chapman
  - Algorithms – 100%
  - Activity diagrams – 100%
  - Created JOIN queries – 100%
- Landon Crispin
  - User Interface: Design and Implementation – 100%
  - Data Structures – 100%
- Dan Ring
  - Design of Tests – 100%
  - Plan of Work – 100%
  - Breakdown of Responsibilities – 33%

# Analysis and Domain Modeling Conceptual Model

## 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, schedule.
5. Database Connection – Verify and establish connection to database ensuring readiness.
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 operator is a valid user or admin
8. Make Appointment – To schedule an appointment via the client application and save it to the database avoiding overbooking.
9. Add user – to allow more login validation users

## 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

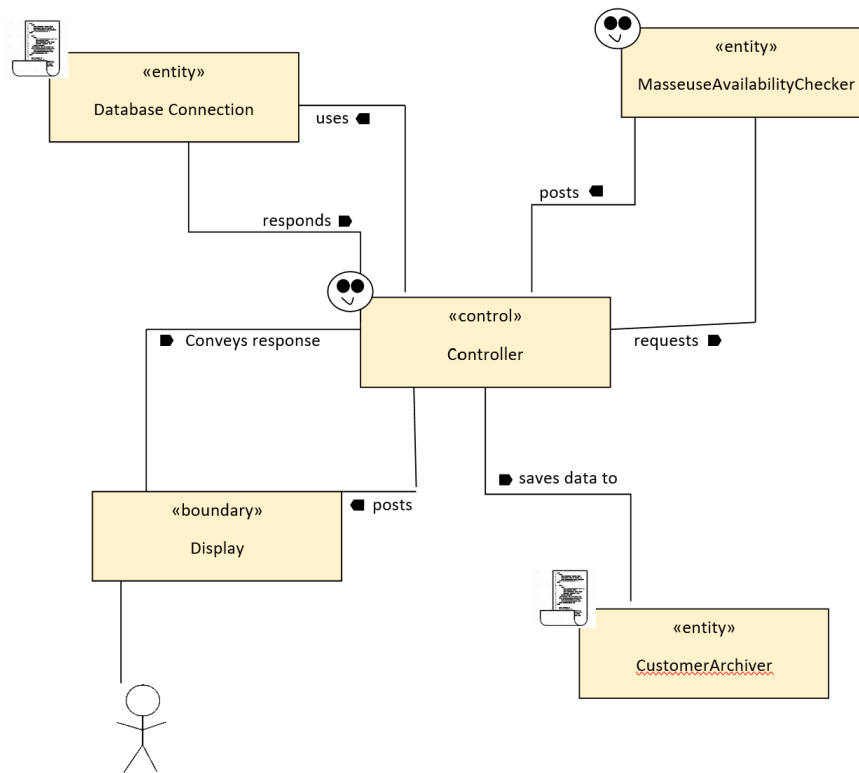
- address – used for customers and masseuses in database.
- appointmentId – integer id of appointment.
- customerId- integer id of customer.
- dbIsVerified – Verification of the connection to database.
- dow – integer indicating days of week masseuse available.
- email – used for customers and masseuses 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.

## Use Case to Domain Traceability Matrix

[illegible]

# Domain Diagram

Domain model diagram for UC-08: MakeAppt



# System Operation Contracts

## Contract CO1: Login

<b>Operation</b>	login(username: string, password: string)
<b>Cross References</b>	UC-01 Login
<b>Preconditions</b>	End user needs to authenticate with the system to access data.
<b>Postconditions</b>	User is allowed access to proceed using the software with appropriate user permissions as assigned.

## Contract CO2: AddClient

<b>Operation</b>	add_client(customer_name: string, address: string, email: string, phone_number: string)
<b>Cross References</b>	UC-02 AddClient
<b>Preconditions</b>	The initiating end user is a valid user or administrator
<b>Postconditions</b>	A new client is added to the SQL database given the provided parameters.

## Contract CO3: AddMasseuse

<b>Operation</b>	add_masseuse(masseuse_name: string, address: string, email: string, phone_number: string, drivers_license_number: string)
<b>Cross References</b>	UC-03 AddMasseuse
<b>Preconditions</b>	The initiating end user is a valid user or administrator
<b>Postconditions</b>	New masseuse added to SQL database

## Contract CO4: MakeAppt

<b>Operation</b>	make_appt(date: string, assigned_masseuse: Masseuse, customer: Customer, service: Service, room_number: int)
<b>Cross References</b>	UC-08 MakeAppt
<b>Preconditions</b>	The initiating end user is a valid user or administrator
<b>Postconditions</b>	SQL database updated with appointment information

## Contract CO5: RemoveAppt

<b>Operation</b>	remove_appt(appointment_id: int)
<b>Cross References</b>	UC-09 RemoveAppt
<b>Preconditions</b>	The initiating end user is a valid user or administrator
<b>Postconditions</b>	Appointment is deleted and available for new appointment.

## Contract CO6: EmailReminder

<b>Operation</b>	email_reminder()
<b>Cross References</b>	UC-12 EmailReminder
<b>Preconditions</b>	First valid user or Admin to logs on each day.
<b>Postconditions</b>	Emails clients with reminder of appointment the next day.



# **Data Model and Persistent Data Storage**

Because a fundamental use case of this software is to create, read, update, and delete several categorizable types of data, it is necessary to have a reliable form of data storage that can accommodate large amounts of data and can access it reliably and quickly.

One of the data storage requirements of this software is to have it persist throughout multiple executions, therefore, the system works with a cloud-based remote SQL database using Amazon RDS. A remote solution to data storage is used so that multiple clients can be in use at the same time and have access to the same data source without having to worry about a form of shared network storage to provide access to the data.

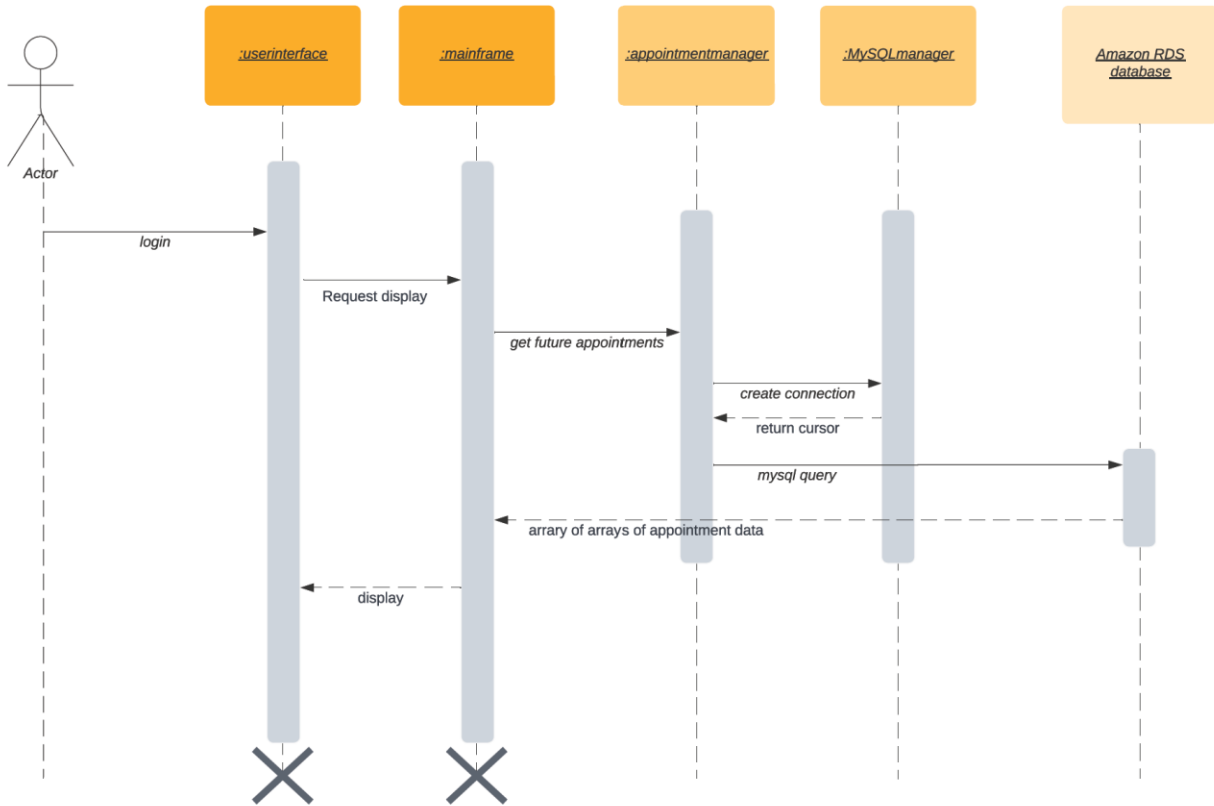
The remote SQL database can very easily be manipulated with the use of Python's native MySQL connector. In using the connector, one can connect to the database with an IP, port, username, and password, then using standard SQL syntax the data can be queried or edited as necessary.

## **Mathematical Model**

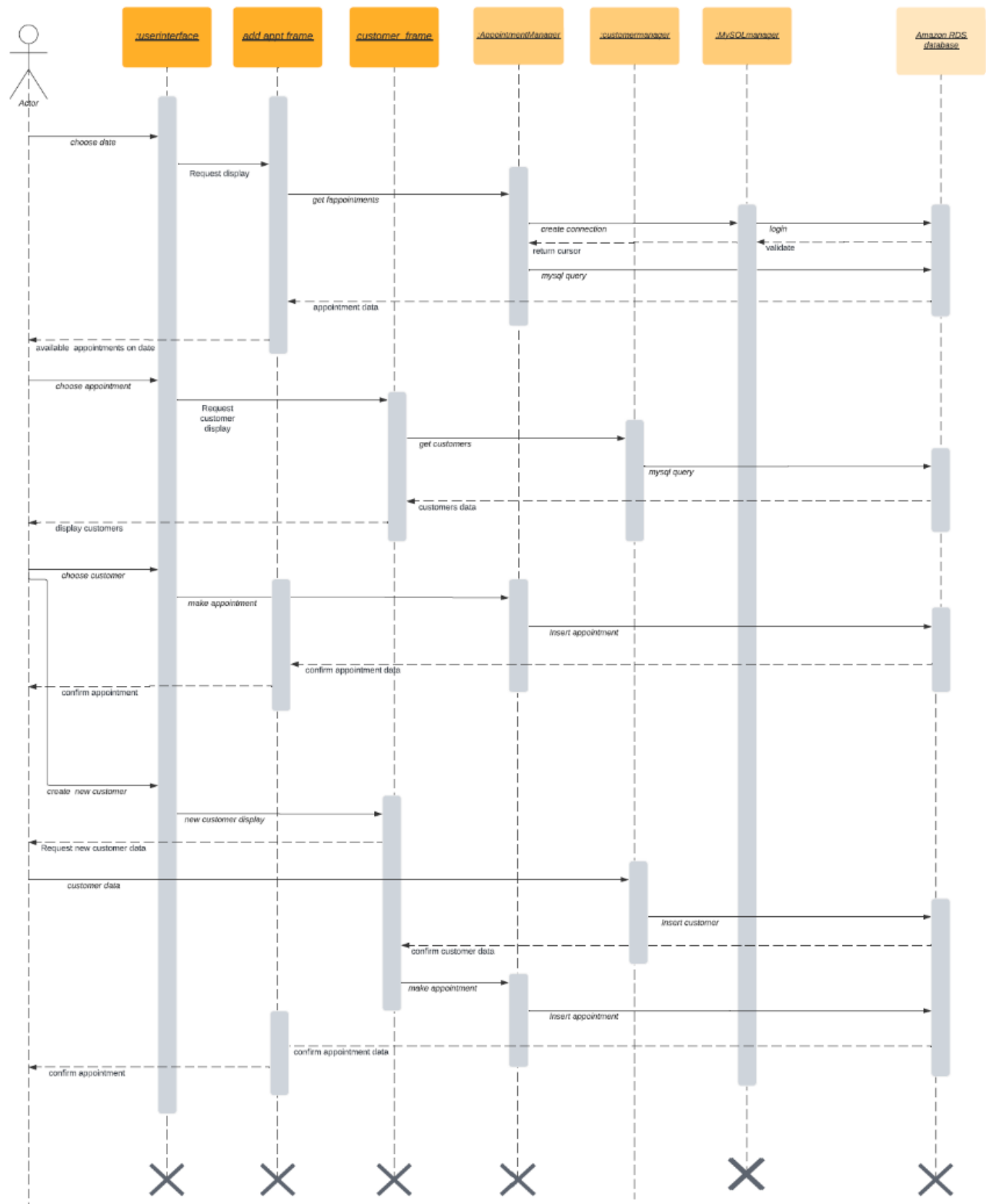
Because the system primary allows the end user an interface to create, read, update, and delete data internal to the company and display them in an easy-to-use format, the project requirements outlined do not require any intensive mathematical operations or algorithms.

# Interaction Diagrams

## UC – 06 Show Schedule



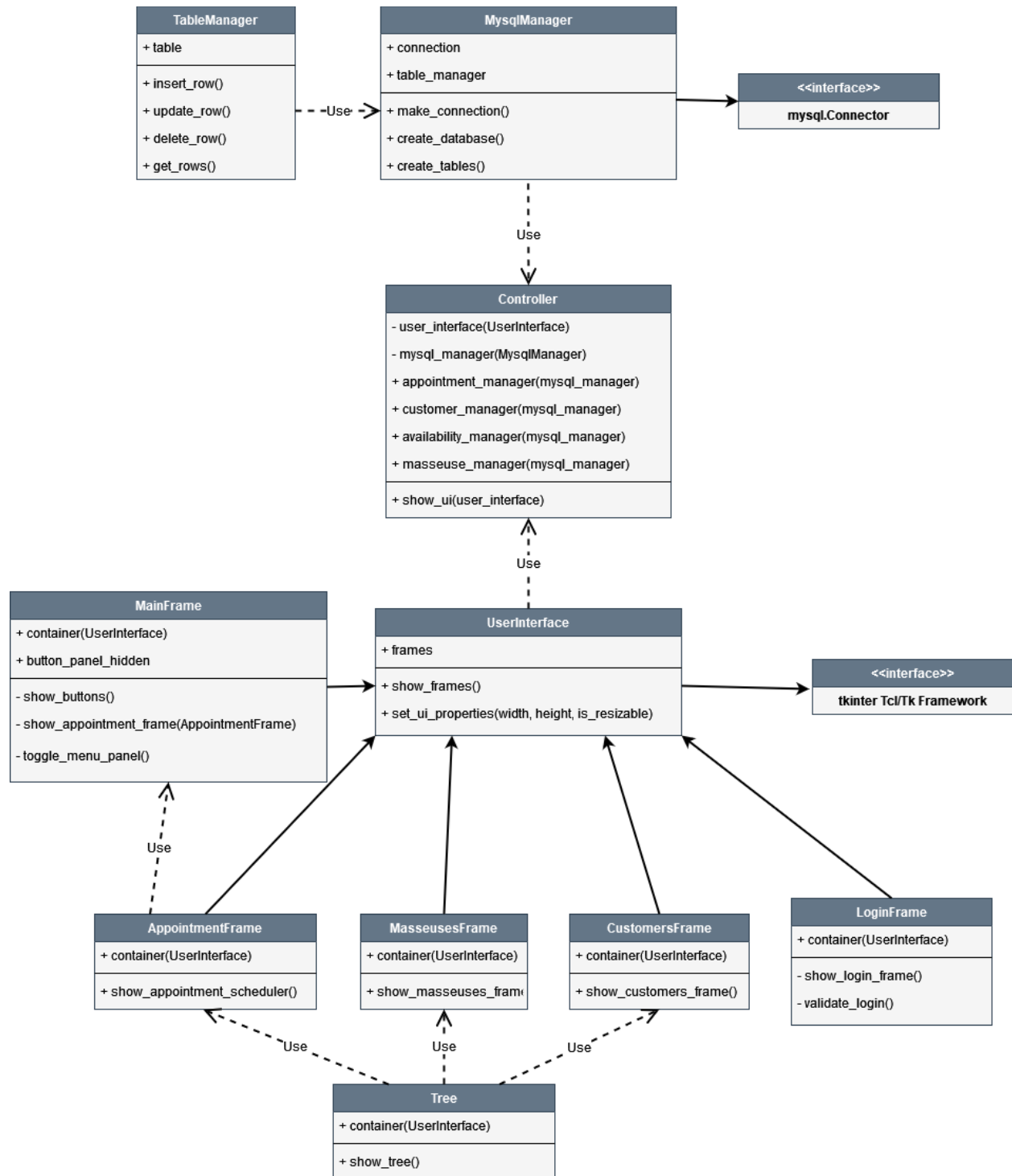
## UC – 08 Add Appointment



# Class Diagram and Interface Specification

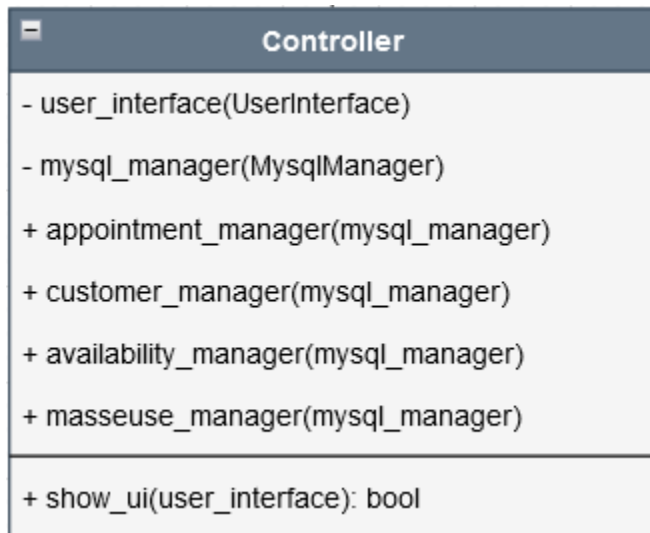
## Class Diagram

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



## Data Types and Operation Signatures

### Controller



#### 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

 UserInterface
+ 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

 MainFrame
+ 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

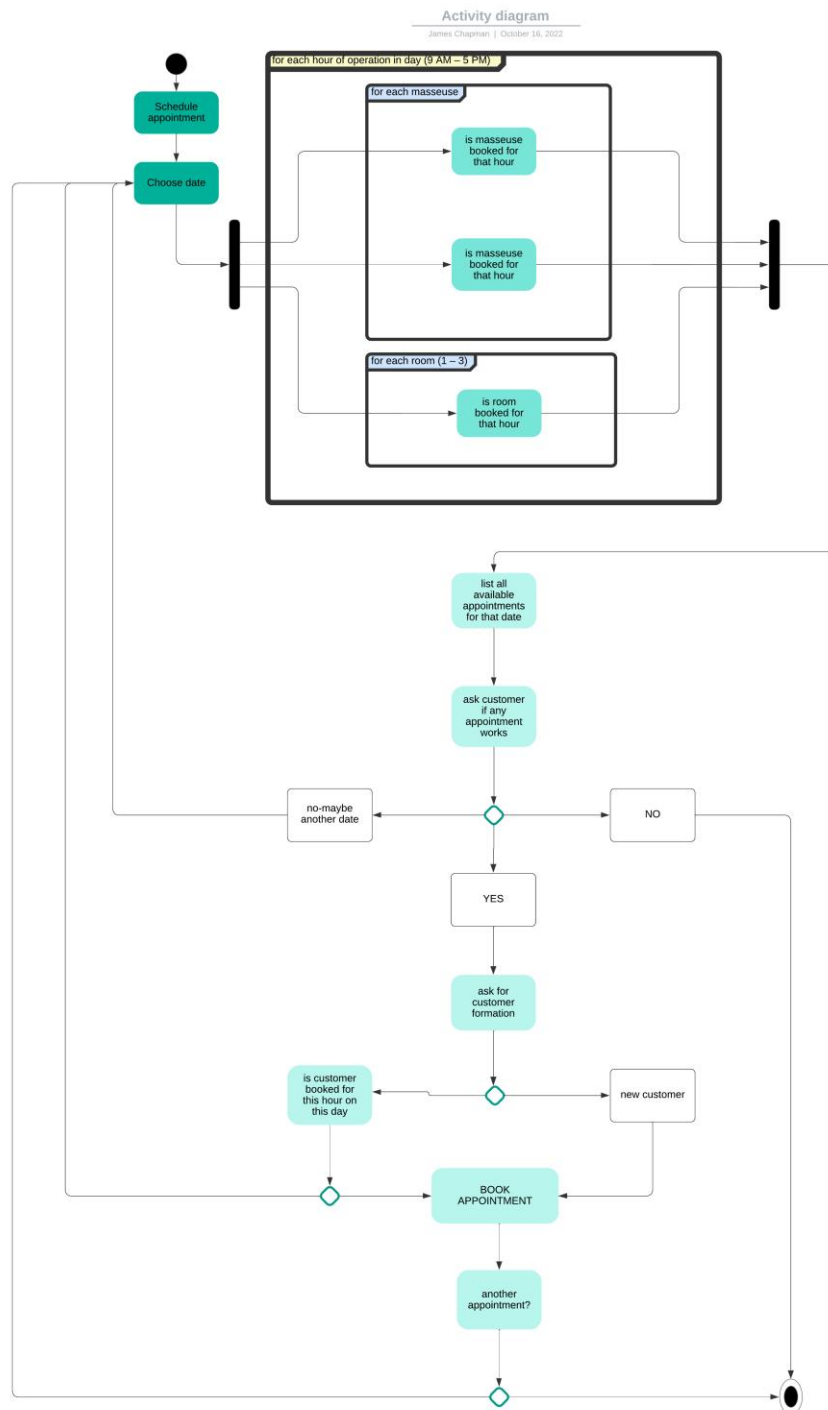
	CD-01	CD-02	CD-03	CD-04	CD-05	CD-06	CD-07	CD-08	CD-09
TableManager	X	X	X		X			X	
MysqlManager	X	X	X		X			X	
Controller	X	X	X	X	X	X	X	X	X
UserInterface	X	X	X	X		X	X	X	X
AppointmentFrame			X	X	X	X		X	
MasseusesFrame		X			X				
CustomersFrame	X				X				
LoginFrame							X		X
Tree	X	X	X	X	X			X	
MainFrame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

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.

# 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.

## User Interface: Design and Implementation

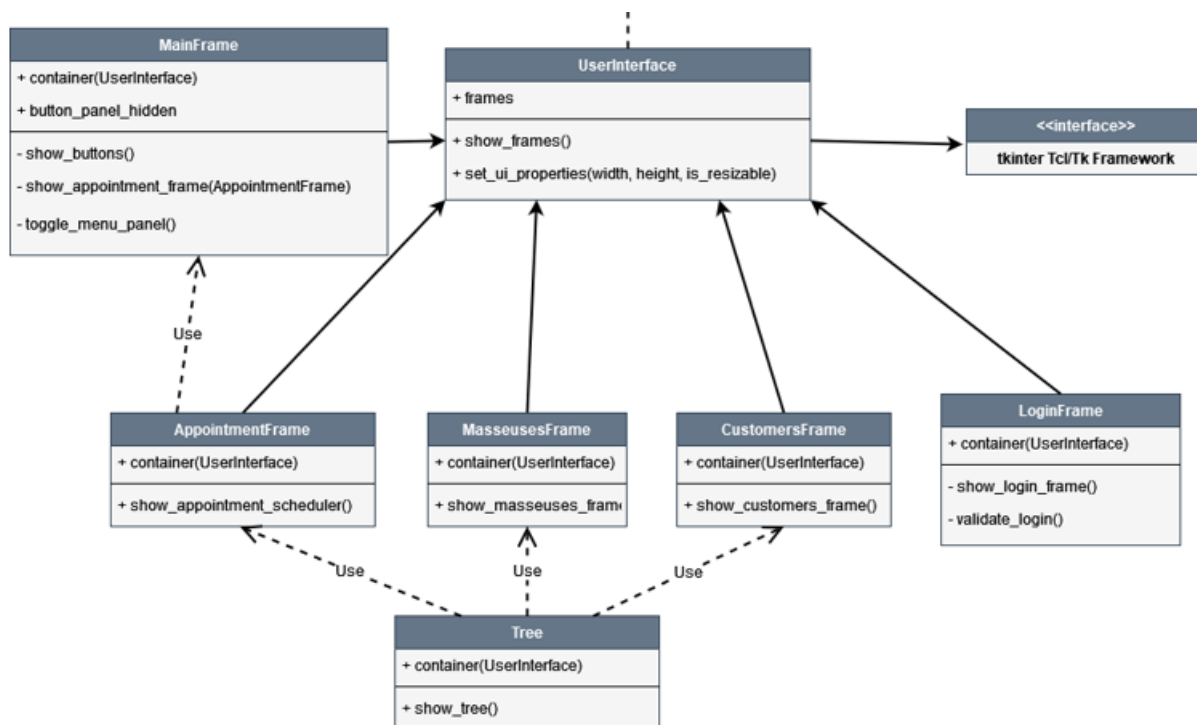
The goal of a user interface should be to allow an end user to utilize the usefulness of a program in a way that is as simple as possible to understand. It's appearance should display information in such a way that the user can view the screen and have few questions about the information presented, besides 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.

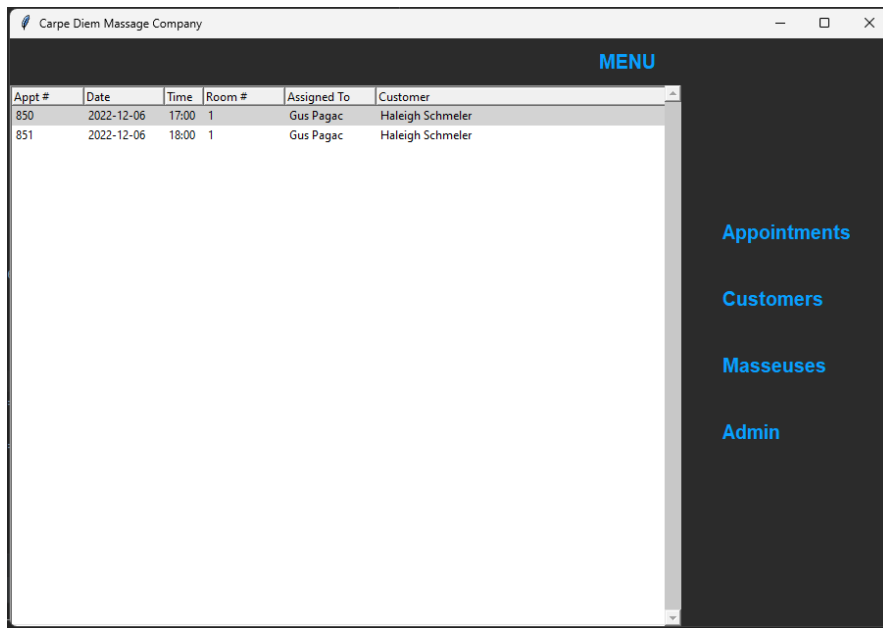
## User Interface Layout

### LoginFrame:



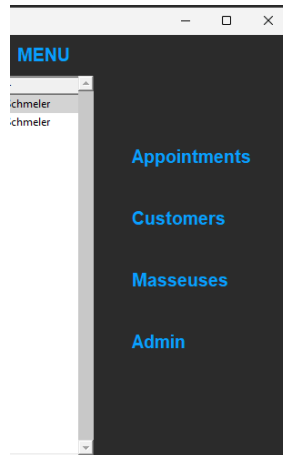
- The LoginFrame is a standalone frame that accepts a username and password as input.
- Login Button:
  - The login button calls a function that queries the current contents of the username and password inputs.
  - The input strings are sent to an authenticator function that checks the strings against 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:



- 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.

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

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

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

## 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.

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>

	3. Press Delete on the MainFrame. 4. Respond Yes to "Are you sure?" prompt pushing the update do the database.
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, MySqlConnection, Controller, MasseusesFrame, CustomersFrame, Tree
Assumptions:	Successfully logged on with database access, and the AppointmentFrame and MainFrame are displayed.
Steps:	1. Press Customers or Masseuses. 2. Select the correct entry from the CustomersFrame or MasseusesFrame. 3. Press Delete on the MainFrame. 4. Respond Yes to "Are you sure?" prompt pushing the update do the database.
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, MySqlConnection, Controller, AppointmentFrame, Tree
Assumptions:	Successfully logged on with database access, and the AppointmentFrame and MainFrame are displayed.
Steps:	1. Press Add. 2. Input appointment information. 3. Select Done to push to the database.
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, MySqlConnection, 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:	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:	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.

# Project Management

## Merging the Contributions from Individual Team Members

Our team has been utilizing shared Office 365 documents which allow us to contribute on our reports concurrently. As a result, we haven't run into trouble compiling our contributions together. Occasionally, if large subsequent changes to the document are occurring, Word will show a merge conflict, but they've been able to be resolved with ease.

Utilizing GIT, James and Landon have been working together to implement the early framework for the database and the UI to get the barebones application up and running.

## Project Coordination Progress Report

Progress thus far can be separated into two categories as the two sub-projects are early in the process of being tied together and implemented:

### **Interfacing between Python's MySQL connector and the remote SQL database.**

1. A mysqlManager module was developed to establish the socket with the SQL database.
2. Using the established connector in the repository, the following use cases are complete.

All functions interface with the MySQL connector and perform a variety of SQL operations:

- a. Appointment – get, update, delete, insert
- b. Masseuse – get, delete, insert
- c. Customer – get, delete, insert
- d. Availability – get, delete insert

### **Designing the user interface using the tkinter Python framework**

1. A UserInterface class was setup to initialize the main frame of the desktop application.
2. Two utility classes for a toolbar in the GUI and a TreeView table were created as well. Both are called by UserInterface on construction of the objects
3. Currently, the GUI main window opens up to the first revision of the appointment scheduler. It displays the appointment entries in a table/list format for the end user.

We are almost ready to integrate basic functionality between the MySQL code and the user interface code to test and fine tune how the data is handled to and from the connectors. Once we have basic functionality, i.e., we get retrieve appointments from the database and display them in the user interface with ease, then we would be comfortable to continue adding additional use cases from the MySQL connector to the user interface and expand from there.

At the time of submission of report one, the current repository version can be found here:

[https://github.com/JamesChapmanNV/CSCI\\_441\\_group\\_11/tree/3df078860656431030a056b8d62046e7f1409fdf](https://github.com/JamesChapmanNV/CSCI_441_group_11/tree/3df078860656431030a056b8d62046e7f1409fdf)

## Plan of Work

1. Start demonstration prep by 10/23/22.
2. Begin implementation forth with.
3. Have UC-08 (at highest priority) operational by 10/23/22.
4. Have functional requirements implemented by 11/1/22.
5. Test units as they are completed.
6. Have GUI fully functional by 11/15/22.
7. Have software debugged and fully documented by 12/1/22
8. Final full Systems tests by End of Semester

## Breakdown of Responsibilities

Dan Ring will continue to work on Python and SQL to start testing the implementations in place and forthcoming by 10/23/22. Work will be started on the Demo part of the project to ensure a smooth presentation starting 10/17/22.

Landon Crispin will continue work on the user interface and implementing its components as planned. Will also work with team members to begin tying together MySQL database connector and the GUI.

James Chapman will continue connecting the database to the interface. Will work on creating new screens for the GUI. Will work on testing basic conditions of functionality. Will work on report 2 part 2.