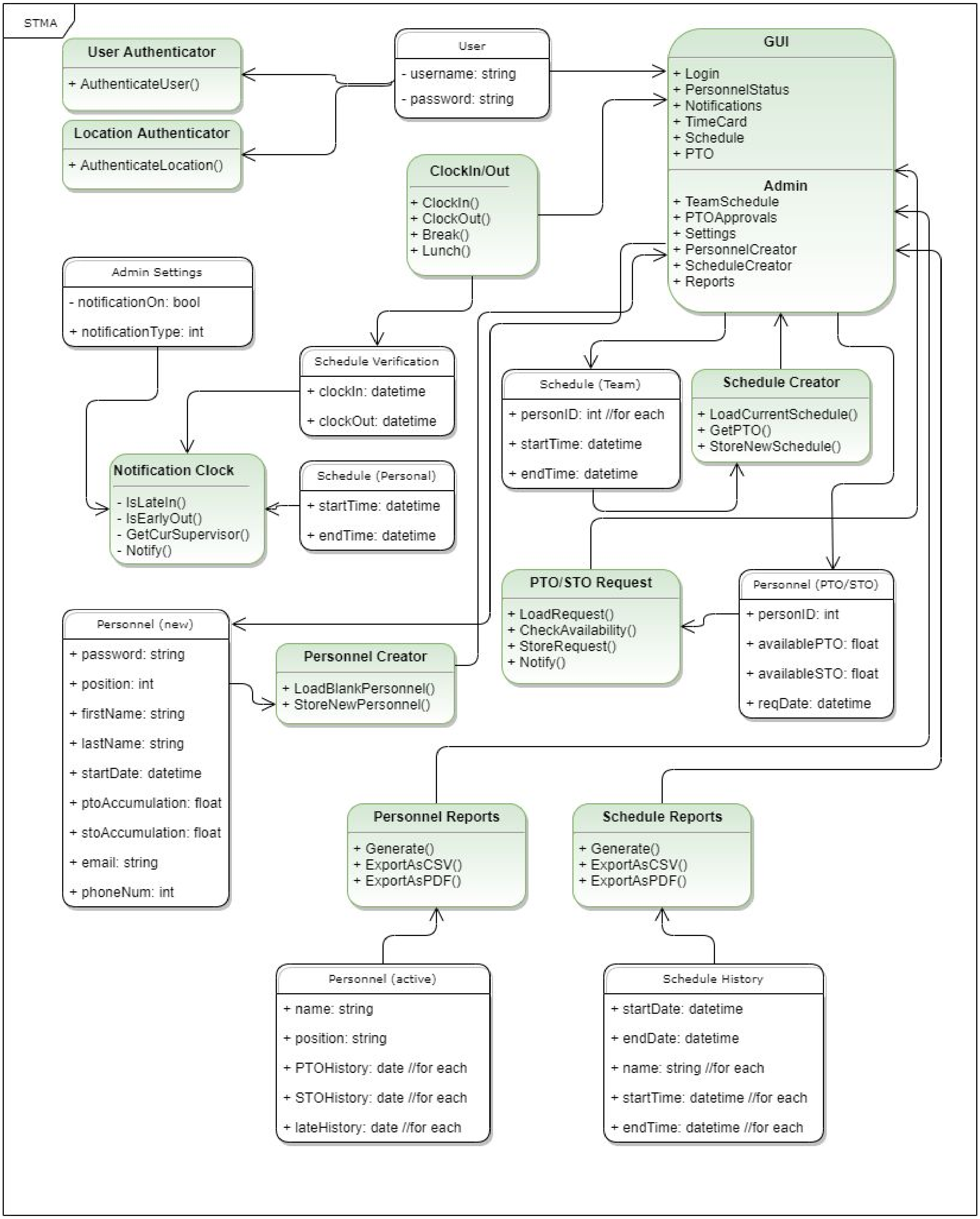
Scheduling and Time Management Application

2019/11/10

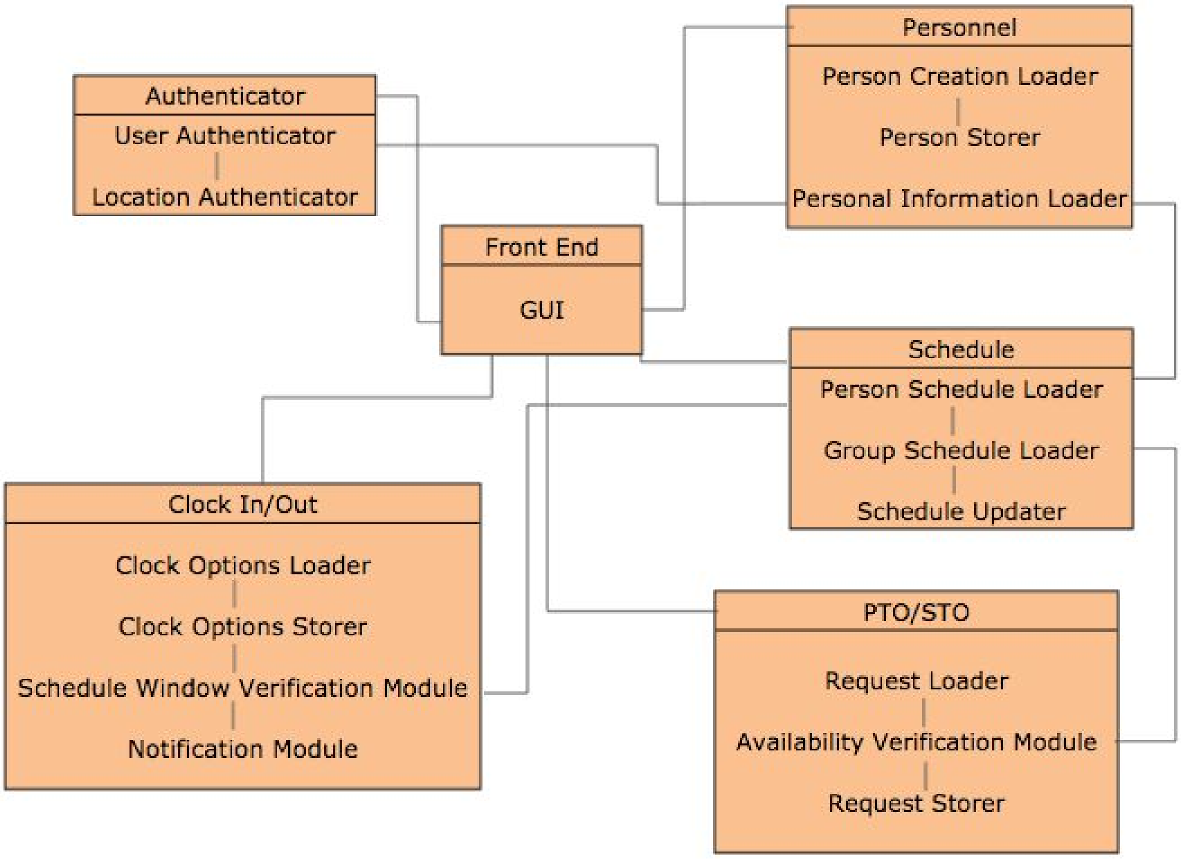
Group 20:
Sunghoon Cho
Michael Czapary
Christopher Elliott
Alex Kolstad
Zachary Wetekamm

UML Class Diagram



Packaging the Implementations

The metrics of coupling and cohesion are applied to evaluate the design complexity of the software modules.



Coupling:

Due to the simplicity of the application, minimal coupling was possible to increase maintainability.

Stamp Coupling

- Personal Schedule Loader will provide structured data to the Schedule Window Verification Module.

- Availability Verification Module will provide structured data to the Group Schedule Loader
- Personal Information Loader will provide information to the Personal Schedule Loader and the User Authenticator.

Common Coupling

- Personal Information Loader and Schedule Window Verification Module both read from the same data.

Control Coupling

- Schedule Window Verification Module calls Person Schedule Loader.
- Availability Verification Module calls Group Schedule Loader.
- Authenticator calls Personal Information Loader.
- GUI calls all modules as if they were submodules.

Cohesion:

Because many modules work as if they were submodules, much cohesion was possible and each module keeps to themselves.

Functional/informational cohesion

- Authenticator only exists to provide user and location authentication.
- Clock In/Out responsibilities independently carry out the function to notify tardy or absent employees.
- PTO/STO loads and stores approved or unapproved requests.
- Schedule module only exists to load and store individual and group schedules.

Communication cohesion

- Authenticator uses same data to authenticate.
- Clock In/Out uses the same data to carry out functions.
- PTO/STO uses same data to load approve and store requests.
- Schedule module uses the same data to load and store individual and group schedules.

Design Assessment

We call for incremental development to be used for the design of this system. All subsections of the system work alone to provide value to one another and each subsection can be built up independently. The major functions of tardy/absent employee notification and request/approval of PTO/STO are based on personal and group schedules. As a result, these functions can be developed on their own to only reference data necessary to do their work. In addition, a logical step of developing the system can start with individual personal profiles. Once the profiles are created, then their schedules can be loaded. With the scheduling complete, the system can then reference it to send alerts based on administrator-set parameters. Approval of PTO/STO can be completed by referencing the individual and group schedules. An update to one section of the system will not require and update to another part of the system and iterative development is unnecessary for the design of this system.

Useful Design Patterns

Facade:
This design pattern would be useful because the GUI acts only as a high level interface with the server that actually does the work. So while the user may use the GUI to send messages or make changes to his or her schedule, the changes are actually being made directly on the server, and being sent back to the client. This is the same way for the push notifications for the mobile client - The mobile client does not send the push notifications, the server sends a push to the app which makes the notification on the user’s device. The whole process is invisible to the user, which is the point of the facade design pattern: For the user to interact with a simple client and have the actual work done on the remote server.

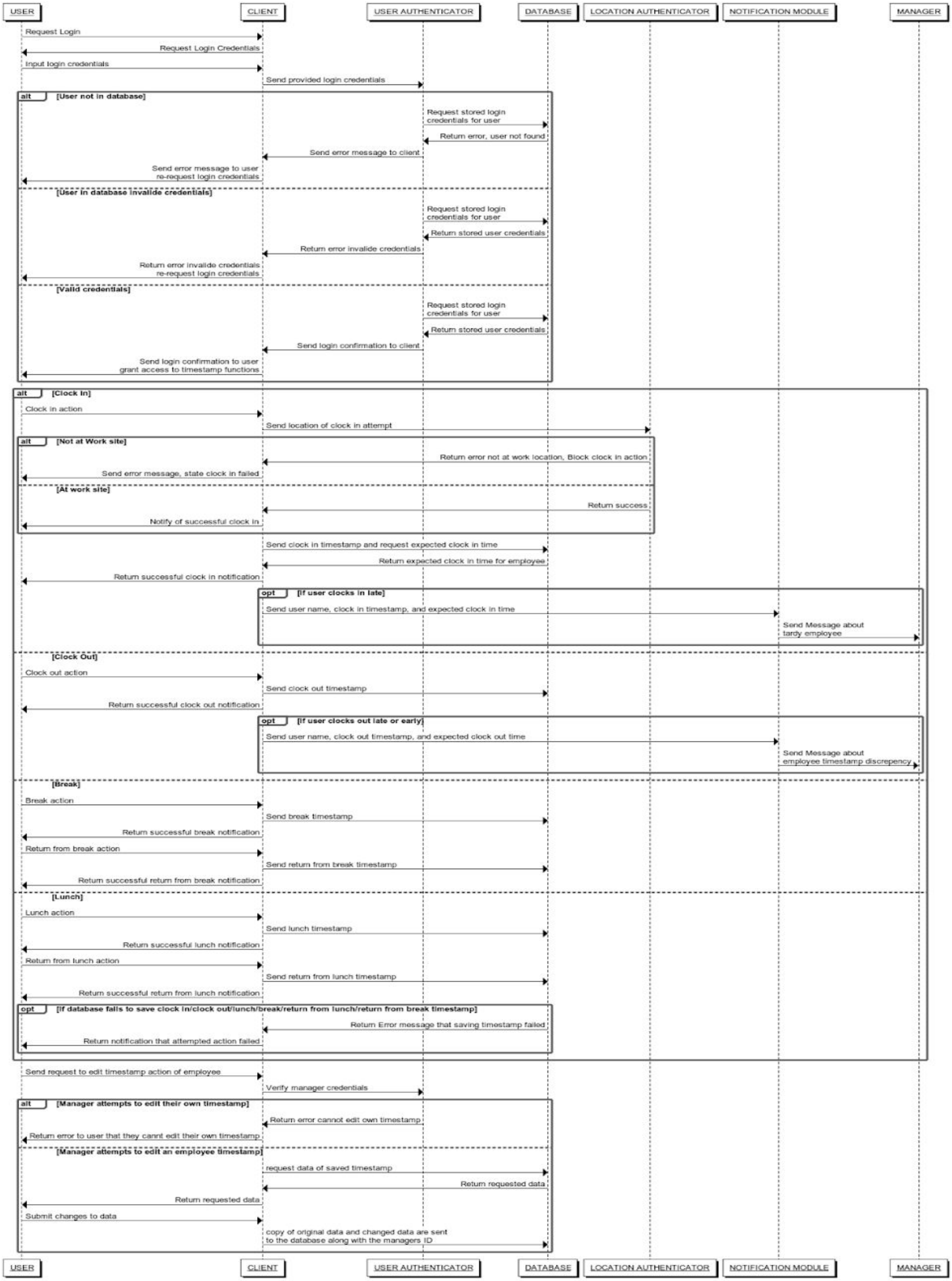
Observer:
The observer design pattern would be useful because one of the most important features of the product is for administrators to get a notification when an employee checks in or is late. This feature requires the capability to look at each employee’s clock

in or clock out status and update when there is a change, which is the essence of the observer design pattern: to watch for changes in an object.

Decorator:
The decorator design pattern would be useful because the functions each user has in the GUI change based on the role given to them. Base employees would only have capabilities pertaining to their own work, whereas administrators would possess the ability to modify other employees' schedules, make decisions like approving or disapproving of base employees' PTO requests, as well as choose to get notifications when employees clock in or are late.

Use Case Sequence Diagram

USE CASE 1 Employee Timestamp Actions


USER
CLIENT
USER AUTHENTICATOR
DATABASE
LOCATION AUTHENTICATOR
NOTIFICATION MODULE
MANAGER
Request Login
Request Login Credentials
Input login credentials
Send provided login credentials
alt
[User not in database]
Request stored login credentials for user
Return error, user not found
Send error message to client
Send error message to user re-request login credentials
[User in database invalide credentials]
Request stored login credentials for user
Return stored user credentials
Return error invalide credentials
Return error invalide credentials re-request login credentials
[Valid credentials]
Request stored login credentials for user
Return stored user credentials
Send login confirmation to client
Send login confirmation to user grant access to timestamp functions
alt
[Clock In]
Clock in action
Send location of clock in attempt
alt
[Not at Work site]
Return error not at work location, Block clock in action
Send error message, state clock in failed
[At work site]
Return success
Notify of successful clock in
Send clock in timestamp and request expected clock in time
Return expected clock in time for employee
Return successful clock in notification
opt
[if user clocks in late]
Send user name, clock in timestamp, and expected clock in time
Send Message about tardy employee
[Clock Out]
Clock out action
Send clock out timestamp
Return successful clock out notification
opt
[if user clocks out late or early]
Send user name, clock out timestamp, and expected clock out time
Send Message about employee timestamp discrepency
[Break]
Break action
Send break timestamp
Return successful break notification
Return from break action
Send return from break timestamp
Return successful return from break notification
[Lunch]
Lunch action
Send lunch timestamp
Return successful lunch notification
Return from lunch action
Send return from lunch timestamp
Return successful return from lunch notification
opt
[If database fails to save clock in/clock out/lunch/break/return from lunch/return from break timestamp]
Return Error message that saving timestamp failed
Return notification that attempted action failed
Send request to edit timestamp action of employee
Verify manager credentials
alt
[Manager attempts to edit their own timestamp]
Return error cannot edit own timestamp
Return error to user that they cannt edit their own timestamp
[Manager attempts to edit an employee timestamp]
request data of saved timestamp
Return requested data
Return requested data
Submit changes to data
copy of original data and changed data are sent to the database along with the managers ID
USER
CLIENT
USER AUTHENTICATOR
DATABASE
LOCATION AUTHENTICATOR
NOTIFICATION MODULE
MANAGER

Interfaces

The interfaces used by the system—Front End Inter-packages, Timestamp messaging, PTO messaging, and database requests—that have precondition inputs and postcondition outputs are as follows:

- Clock-In/Out Messaging Interface
  - Input: Validated timestamp
  - Output: Personnel data (name, timestamp data) as a notification to appropriate supervisor (if timestamp is flagged).
- Time Off Messaging Interface
  - Input: Time off request form data, time off request approval or rejection.
  - Output: Time off request data notification to appropriate supervisor, time off request result to the employee who submitted it.
- Schedule Interface
  - Input: Schedule request from client GUI, personnel data, time off data, clock in/out data.
  - Output: Requested schedule data to client GUI
- Database Interface
  - Input: Login credential request from the Authenticator, employee timestamp from the client, time off request data, schedule data, personnel data.
  - Output: Stored user credentials to Authenticator, timestamp of employee to the client, time off data responses, schedule data responses, personnel data responses.
- Computer Location Services
  - Input: System call to get current location data.
  - Output: Current location coordinates (to validate user is at work location)

Inter-package Interfaces:

- Front End (Web GUI)
  - Input: Actions from the user (requests for clock-in/clock-out, time-off, schedule, login/logout, personnel information).
  - Output: View responses to web GUI with success or failure action results.
- Authenticator
  - Input: Username credential, password credential, indicator if or if not an admin/supervisor, clock-in and clock-out timestamp, time-off request, location data

- Output: Validated data from the database request (will be allowed or denied depending if admin/supervisor).

Exceptions

The system only has two components, and thus will have two types of exceptions it will have to handle. Each type of exception it will have to handle is detailed below with more specific type of exceptions covered under the broad two definitions.

Internal System Exceptions:

These errors would be errors with the code itself and must be handled gracefully by the program's own error handling techniques based on the error occurred. For instance, the system must be able to handle types of invalid input, or even nonexistent data. An employee may try to apply for PTO after they have used all of their available PTO days, which would trigger an exception forcing the user to instead apply for special use case time off, which has a different application page. No error should be able to break the client, and should be able to self recover. Entering nonexistent data should prompt the user to enter the data again, until valid data is received, as should entering invalid data. Checks should be in place for invalid dates when applying for time off or choosing users' schedules that would prompt the user for a valid date.

External System Exceptions:

These errors would be the fault of the connections between the client and the server. When such an error occurs, the client should continue trying to connect to the server until a specific timeout period has been reached, where the client will alert the user to check their internet connection or notify the user that the server is offline. Similarly, if system calls start failing, the program should notify the user that the system has become unresponsive, or that the program may need administrator privileges to run properly

Team Member Contributions

- Group meeting (all members attended) via Google Hangouts on Wednesday 11/6 to discuss UML design, packaging of implementations, necessary interfaces and design patterns. Group communication and updates throughout the week via Slack.
- Christopher Elliott: UML class diagram.
- Sunghoon Cho: Packaging the implementations; design assessment.
- Alex Kolstad: Useful design patterns; exceptions.
- Michael Czapary: Use case sequence diagram.
- Zachary Wetekamm: Interfaces; document assembly.

Each group member was proactive in the group discussion to ensure each portion remained relative. Communication through slack and peer-review of documentation contributions were made by each group member.