OOAD Project Part 4: Progress part 2

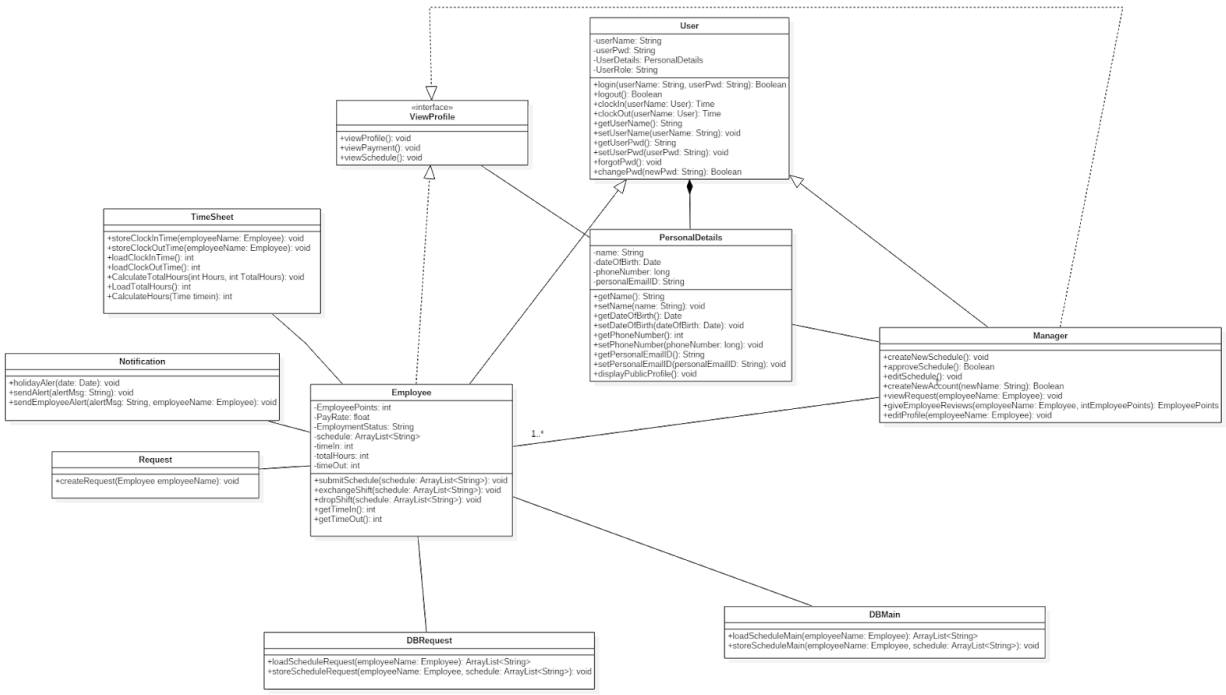
Title: Work Force Management

Team number: 52

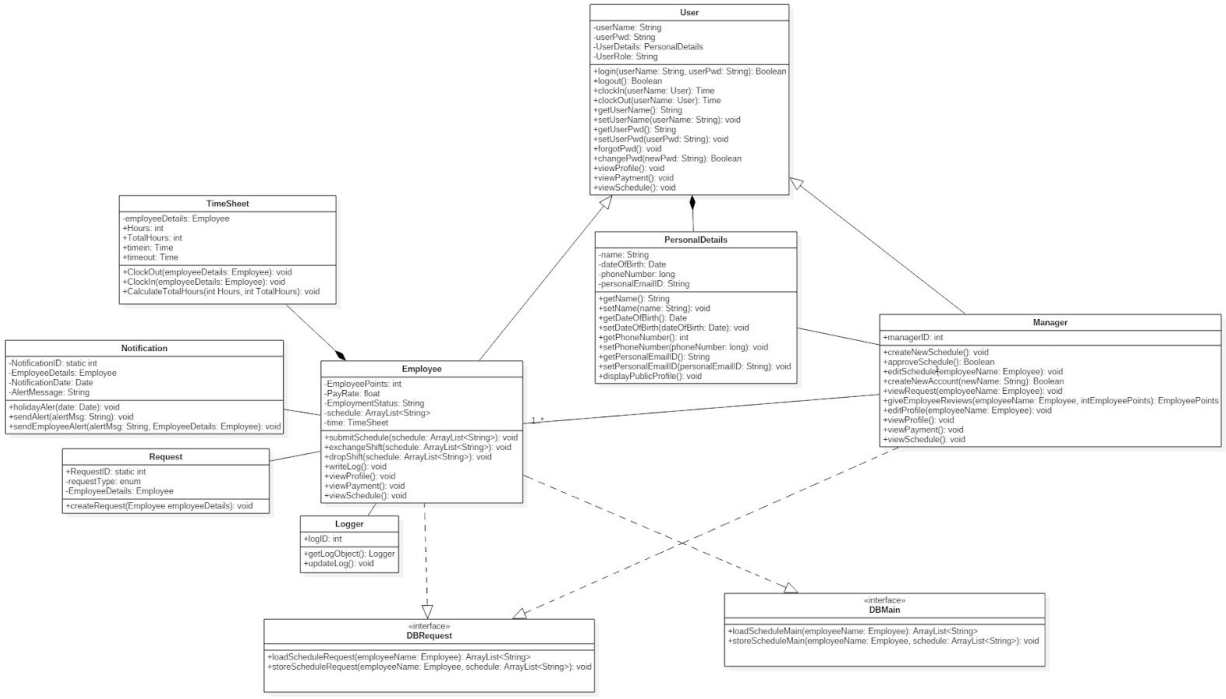
Project members: Aswath Gundepally
Omkar Prabhu
Vipraja Patil

Description: A web based application that helps Management and Employees with schedule management, initiating and approving shift change requests and time-offs, and managing payment details for an organization.

Previous Class Diagram:



Updated Class Diagram:



Summary: Added new design patterns. A Singleton pattern is used for the logger class. A log number is incremented upon every entry to the log file. This helps with a functional requirement. Also modified timesheet class to reduce code duplication and unnecessary function calls which were there previously.

Breakdown:

Vipraja Patil:

Created updated class diagram

Added Manager class

Added Template design pattern

Ashwath Gundepally:

Integrated all the frameworks(spring, maven, tomcat, etc)

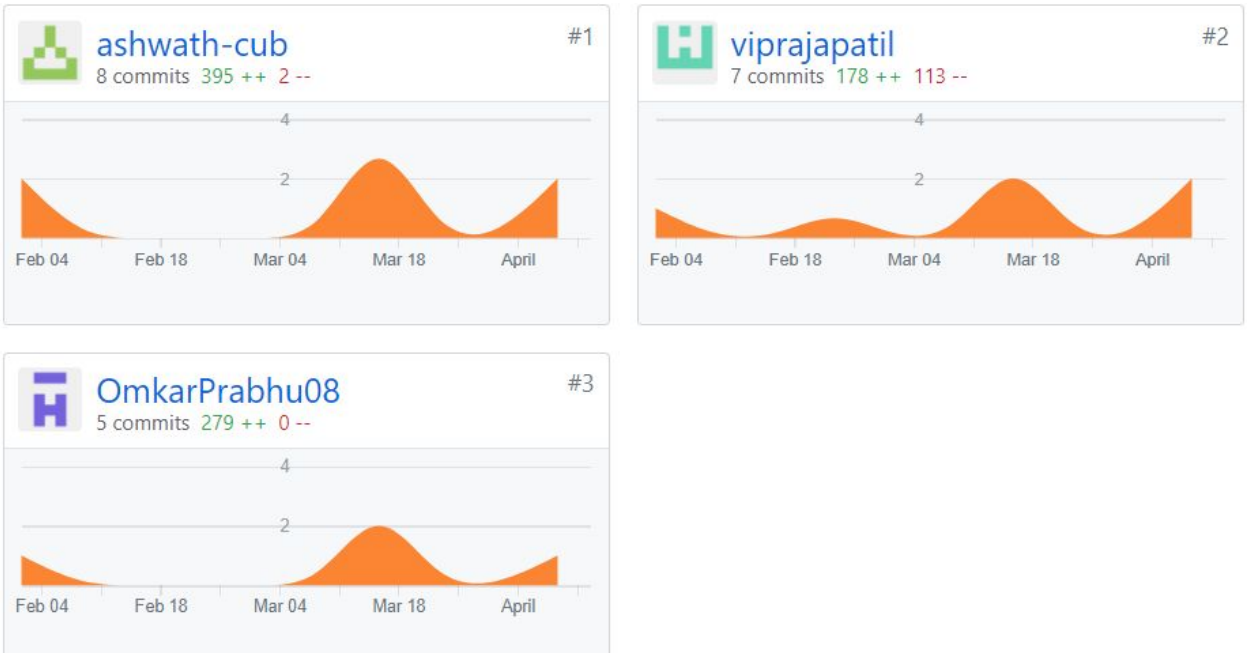
Implemented part of login

Omkar Prabhu:

Added design pattern for Singleton

Added Logger class

GitHub Graph:



Estimate Remaining effort:

A week's worth of coding is needed to implement the entire design as presented in part 2. End to End implementation of use cases is remaining.

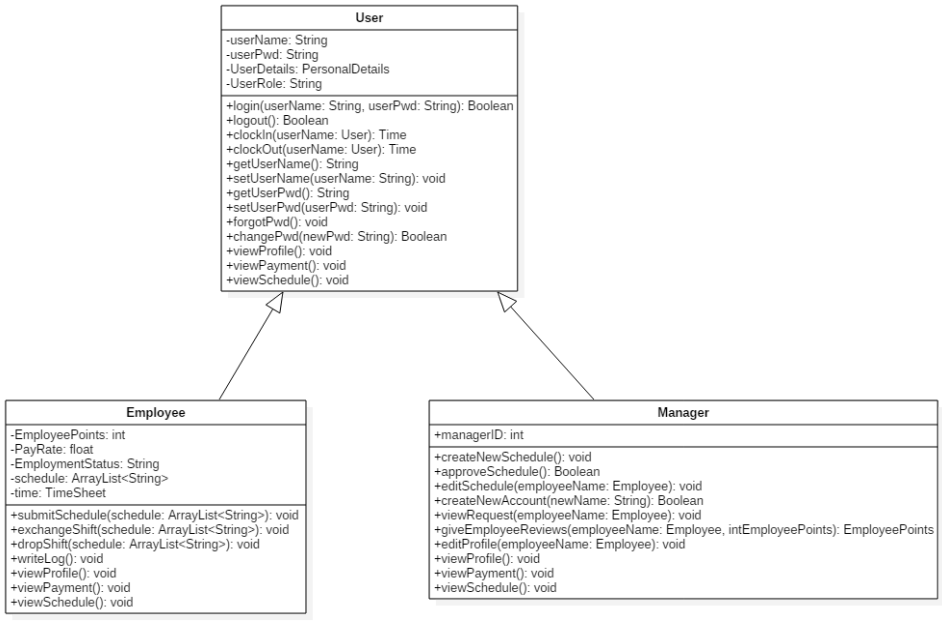
Progress has been made in adding design patterns and refactoring the code. The class diagram is changing with every iteration to accommodate new design patterns and optimizations.

Design patterns:

1. Template Design Pattern

The Template method design pattern lets subclasses implement varying behavior (through method overriding). In class diagram, we are using template method in classes User, Employee and Manager to avoid the duplication of code. Which controls at what point(s) subclassing is allowed. As opposed to a simple polymorphic override, where the base method would be

entirely rewritten allowing radical change to the workflow, only the specific details of the workflow are allowed to change.

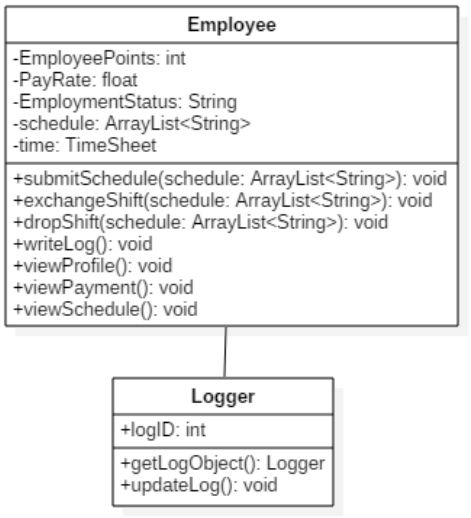


2. Singleton Design Pattern

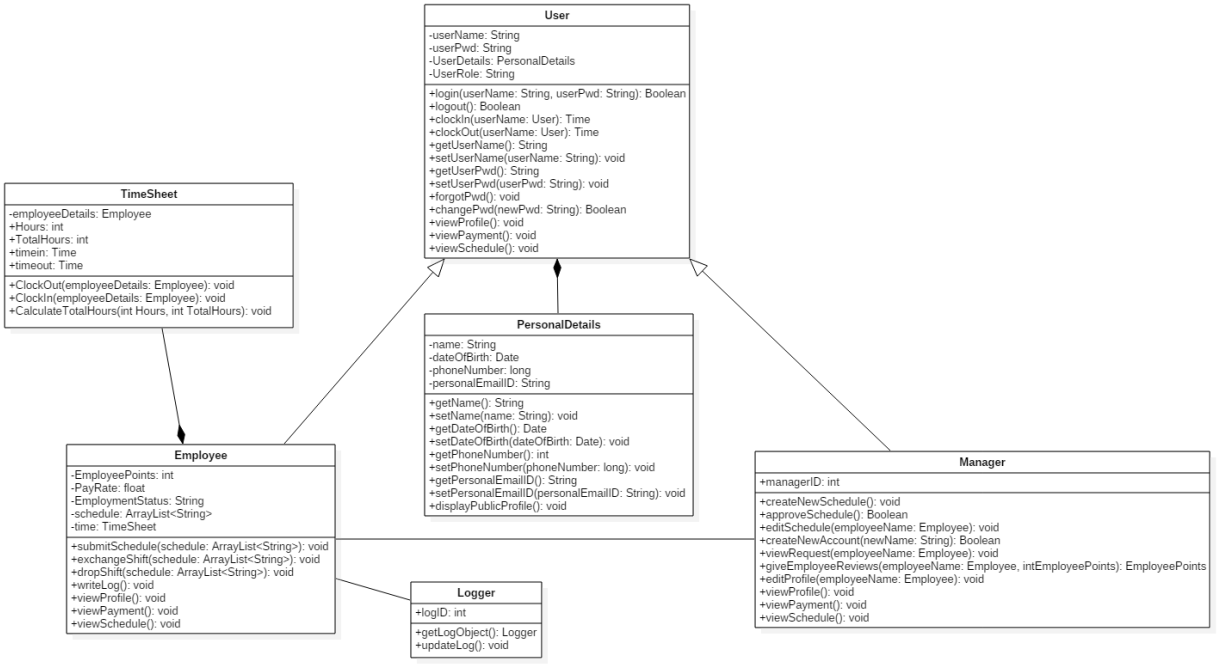
We are planning to use Singleton to implement the Logger because for every instance of the employee class, there needs to be only one instance of the Logger class that has a unique Log ID.

Also, all the instances of the Employee class are going to be writing to a single (and same) log file in a sequential manner that is they are using shared resource. Use of static methods is also an option, however, the code becomes harder to extend and test. This further reinforces the need for using the SIngleton Design Pattern.

The other reason to use singleton pattern while implementing logger is direction of dataflow,it’s only in one direction. Since we are not going to use the data again we don’t have to care about race conditions and thus using mutexes.



Implemented Class Diagram:



Final Iteration:

- Standard Interface to database
- End to End Implementation of all use cases like request time-off, exchange shifts.
- And some refactoring to optimize the code with design patterns one last time.
- Some test code can also be added for verification.