School Management System Final Report

Joel Barkley
Dmitriy Tarasov
Souneth Ly

1.

Features implemented:

The following Business Requirements were implemented:

ID	Requirement	Topic Area	Actor	Priority
BR-001	Limit access to non-school users	Login	Deans, Teacher, Student	High
BR-003	Teachers review	Reporting	Deans	Medium

The following User Requirements were implemented:

ID	Requirement	Topic Area	Actor	Priority
UR-001	Login	Login	Student,Dean,Teacher	Critical
UR-002	See/pay tuition rates	Tuition	Student	High
UR-003	Add teacher	Teacher	Dean	Medium
UR-006	Set Grade	Classes	Teacher	High
UR-007	See Grades	Classes	Student	High
UR-009	Create Teacher Report	Reporting	Dean	Low
UR-010	View Teacher Report	Reporting	Teacher	Low
UR-011	Pay Tuition	Billing	Student	High

The following Non-Functional Requirements were implemented:

ID	Requirement	Topic Area	Actor	Priority
NF-002	Reliability	Data Persistence	System Admin	Med

2.

Features not implemented:

The following User Requirements were *not* implemented:

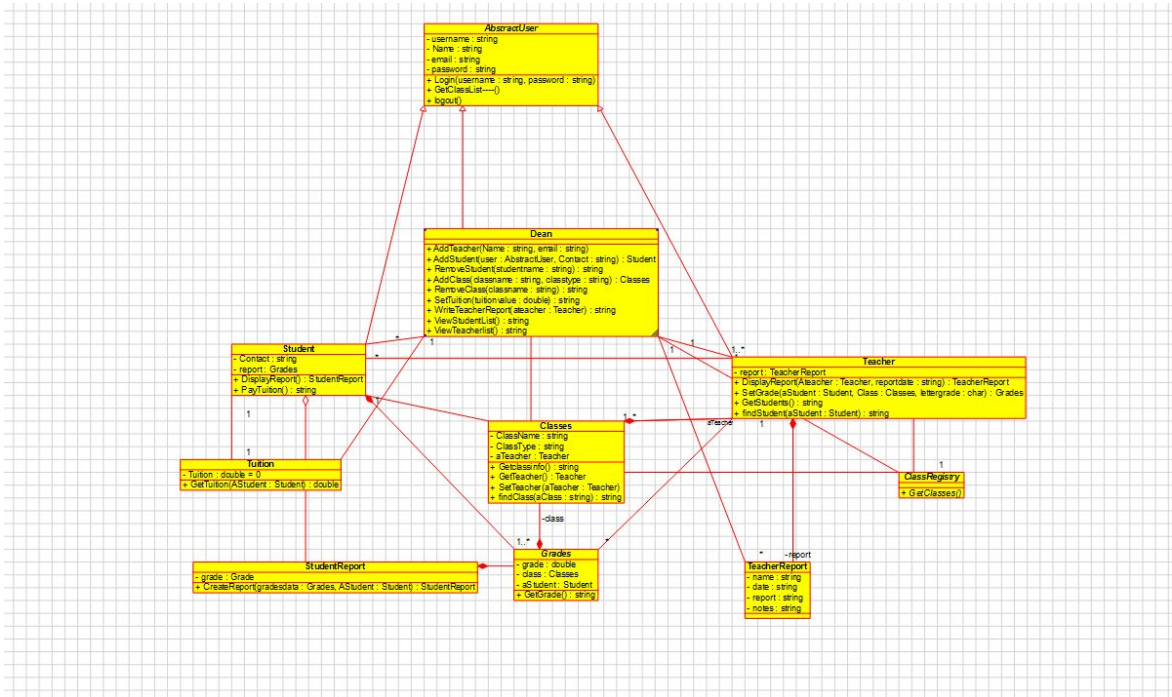
ID	Requirement	Topic Area	Actor	Priority
UR-004	Add Class to Class List	Classes	Dean	High
UR-005	Remove Class from Class List	Classes	Dean	Medium
UR-008	View Class List	Classes	Dean/Teacher	High

The following Non-Functional Requirements were *not* implemented:

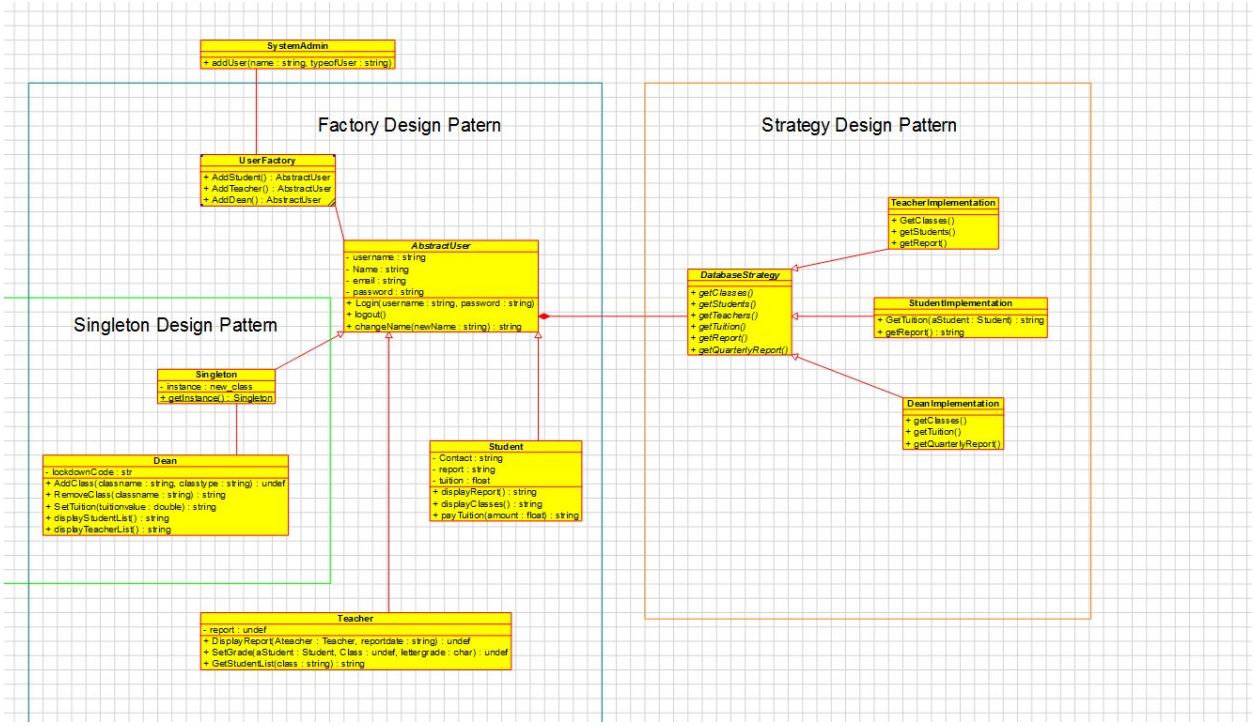
ID	Requirement	Topic Area	Actor	Priority
NF-001	Billing Security	Security	Student	Low
NF-002	Reliability	Data Persistence	System Admin	Med

3.

Part 2 Class diagram:



Part 3 Class diagram:



During the refactoring phase of this project, our class diagram changed significantly. First, we tried to simplify the class diagram wherever possible, so that it became less of a tangled mess. We also changed where some methods were located -- originally we tried to base what class operations were located based on permissions, but afterword changed it to better follow design principles -- encapsulation the attributes and operations that an object should do in a single class. We also tried to implement a few design patterns in our later class diagram. We added a userfactory to handle the runtime creation of users, that are initiated by the dean. We also added a strategy design pattern for database calls -- the idea was that different users might want to make a similar database call, but expect different results depending on the type of user.

4.

Design patterns:

The database strategy part of the diagram vanished from the model as the functionality of that was done by a database.The strategy would of been a good system of managing data if we had implement our own database read and write instead of SQL. We used the factory design pattern for the creation of new instances of students and teachers. The factory design pattern is a good choice as the type of user being created wouldn't be known until run time. Singleton was also

implemented, as a check in the userfactory -- there is only allowed to be one instance of a dean, and if there was a dean already saved in the database, then the user factory would effectively ignore the request to make a new dean.

5.

What We've learned:

This project was definitely a huge learning experience -- especially regarding the usage and implementation of spring and hibernate. In fact, getting both of these set up, and understanding how they worked probably took the longest time in this project. We also learned that analysis and design does not stop once the class diagram is made -- sometimes the class diagram fits well for the actual implementation, but other times this isn't the case. This can be seen in the two design patterns we tried to implement -- the Factory pattern and Strategy pattern. The factory pattern was not too hard to implement, and worked perfectly for our needs in this project. However when trying to implement the strategy pattern, and get the database up and running in general, we learned that there is actually a set pattern for database calls for different object, where you create a data access object layer that works directly with jdbc and hibernate, and an object service layer, which is what the controller would use to retrieve object information. At the time we designed that design pattern, we had no idea there was already a commonly used pattern to achieve what we were trying to do, and thus we scrapped our initial pattern to use the standard pattern. However, the upfront development of the model made the creation of the model quite simple. The result of the project would of been better if there was concurrent coding development and design development. This would resulted in concurrent domain knowledge of the problem influencing the design of the project.