Work Force Management

Final Report

**Team Number:** 52

**Team Members:** Ashwath Gundepally
Omkar Prabhu
Vipraja Patil

**Vision:** Work Force Management is a shift and schedule management app designed with the an intent to minimise the usage of email for shift and schedule management at the workplace.

**Actors:** Employee, Manager, Admin

**Description:** Using this app, the employees can view public profiles of all the employees involved, submit weekly schedules, create shift and drop requests. Management can acess employee information, add or remove employee and approve of the submitted schedules.
The app is written in Java and the front end is written in JS. It uses the Spring MVC with Hibernate for ORM.

**List of features implemented:**

| **USER REQUIREMENTS** | | | |
|---|---|---|---|
| **ID** | **Actor** | **Requirement** | **Priority** |
| US-01 | Employee, Manager | As a user, I should be able to login | High |
| US-02 | Manager | As a manager, I should be able to create employee accounts | High |
| | | | |
| US-04 | Manager | As a manager, I should be able to edit employee schedules | High |
| US-05 | Employee | As an employee, I should be able to view public profiles of other employees | Medium |
| US-10 | Employee | As an employee, I should be able to view payment details | Low |
| US-11 | Employee, Manager | As a user, I should be able to clock-in and clock-out | High |
| US-14 | Employee, Manager | As a user, I should be able to view | High |

| | | | |
|---|---|---|---|
| | | schedules | |
| US-15 | Manager | As a manager, I can view requests for shift exchange, and timeoff | High |
| US-16 | Manager | As a manager, I can review Employee’s performance | Low |

| **BUSINESS REQUIREMENTS** | | |
|---|---|---|
| **ID** | **Requirements** | **Priority** |
| BR-01 | Maximum 100 employee accounts can be created | High |

| **FUNCTIONAL REQUIREMENTS** | | |
|---|---|---|
| **ID** | **Requirements** | **Priority** |
| FR-01 | Email the employee the link to reset their password | High |
| FR-02 | Email the employees about holidays | Low |
| FR-03 | An alert should be send to the manager if the employee doesn’t turn up for their shift | Medium |
| FR-04 | Email the employees about their performance status | Low |

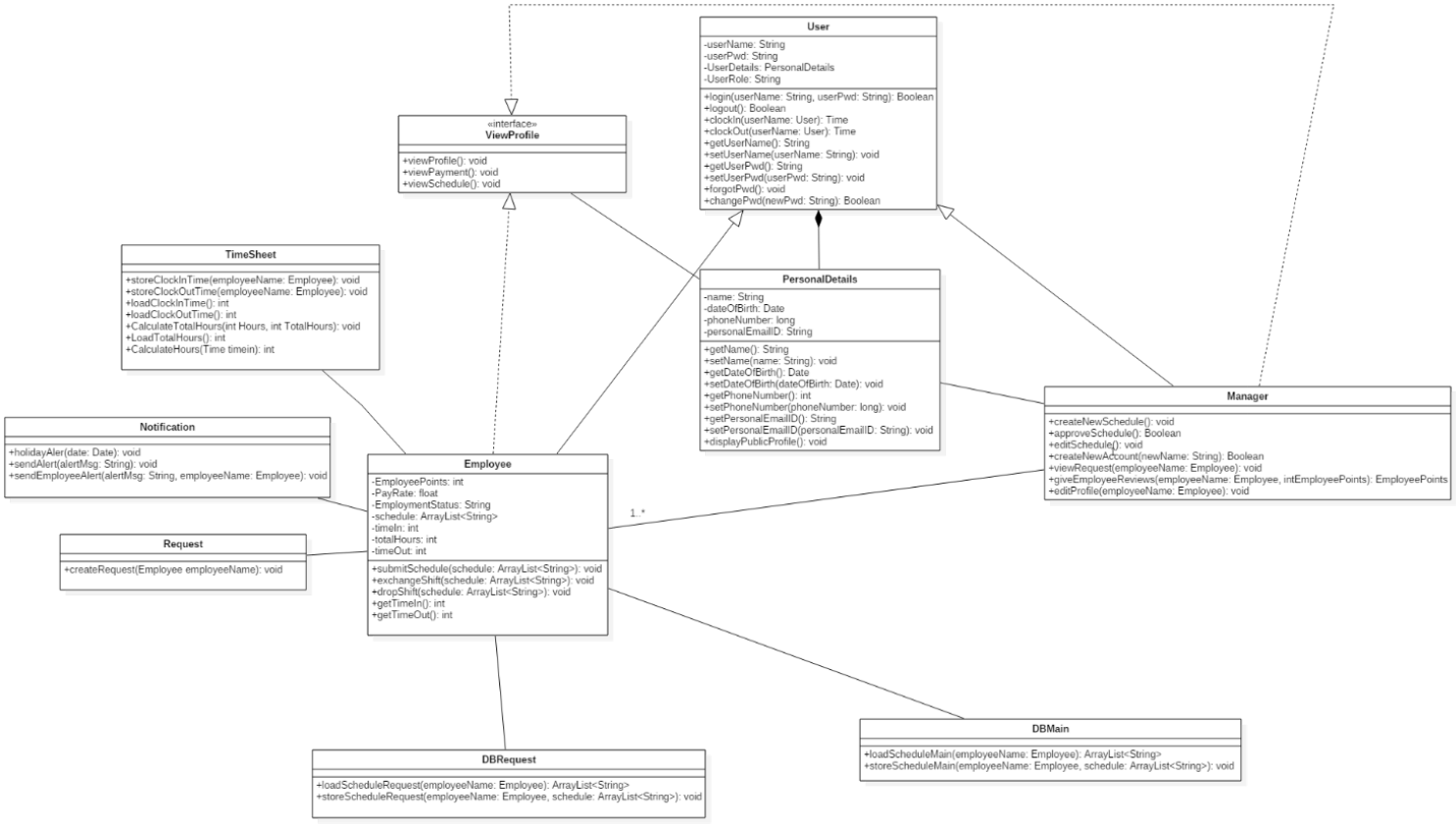
| **NON-FUNCTIONAL REQUIREMENTS** | | |
|---|---|---|
| **ID** | **Requirements** | **Priority** |
| NFR-01 | Store salted hash of the password in the database | Medium |
| NFR-02 | If the attempts to login exceeds more than 5 | Low |

| | | |
|---|---|---|
| | times, the system will block the user account | |
| NFR-03 | The system should be able to handle 20 concurrent employee logins | Low |

**List of features not implemented:**

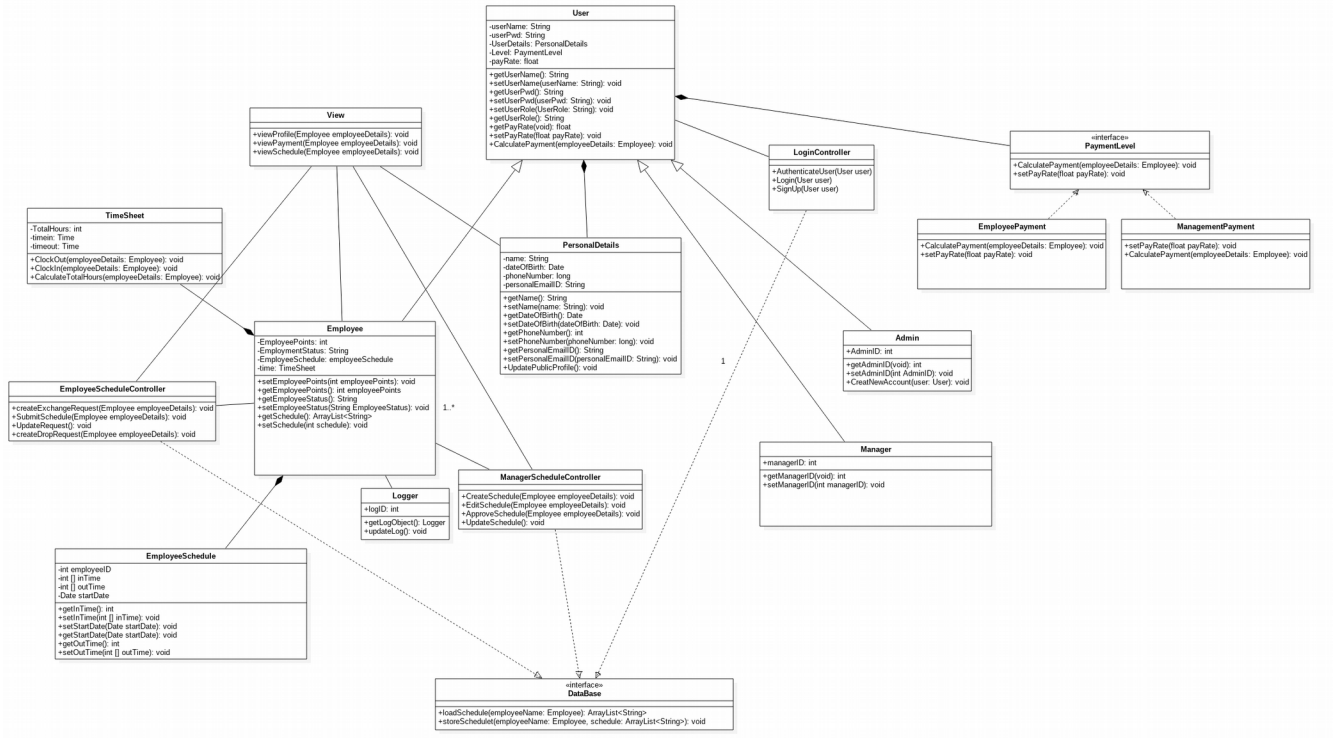
| ID | Actor | Requirement | Priority |
|---|---|---|---|
| US-03 | Manager | As a manager, I should be able to create employee schedules | High |
| US-06 | Employee | As an employee, I should be able to request to exchange my shifts | Medium |
| US-07 | Employee | As an employee, I should be able to request to drop my shifts | Medium |
| US-08 | Manager | As a manager, I should be able to approve of leave/ shift exchange requests | Medium |
| US-09 | Manager | As a manager, I should be able to edit details of the employees | High |
| US-12 | Employee, Manager | As a user, I should be able to change the passwords | High |
| US-13 | Employee | As an employee, I should be able to submit my availability for the upcoming work week. | Medium |

## Class Diagram Part 2:




.

**Class Diagram 5:**




**What changed and why?**

Primarily, we have now adopted the MVC framework which was missing in part 2. The MVC provides us with a convenient way of isolating the Model(data), Controller(control) and View(front end) of a project in a simple and efficient manner. The controller also helps with acting like an interface between the Model and the View effectively.

**Strategy Design Pattern**

We have two methods named as SetPayrate and CalculateTotalPayment and implementation of these methods are depending on what kind of users they are depending on the allowances and taxes and the checks in setter method

The method is the same for admin and the employee class, but different for the employee class.

In our system there are two methods set pay rate and calculate payment and implementation of these methods is dependent on what level of employee the user is.

Explaining further the setter method will have different checks depending on employee level and calculate payment methods would be different depending on allowances they get and taxes

In our case the manager and admin are at same level whereas employees are at different level thus they have same implementation but different than what employee class have

So we can't define an abstract method in user class and even if we define them individually in all classes or introduce interface for users at same level there would be a problem of code duplication

This problem would grow further as we introduce users at new level

Thus we are using strategy design pattern over here where we have introduced a behaviour name userlevel and defined that as an interface, Where as the concrete classes using that interface would have actual methods

Now in this case the problem of code duplication is solved since we have to write actual method in only one class and using polymorphism depending on attribute value of the level the classes would use the methods

Also since the interface is there we can add many other classes without changing client code

**Singleton Design Pattern**

The singleton design pattern is used to implement the logger. This pattern is helpful because one instance of the logger class runs throughout the execution logging messages from multiple sources. It provides a global point of access for any log source and the pattern makes for multiple sources to register to it easy.

Designing up front did certainly help, it gave us an initial direction and a start to systematically approach a project of this type.

We have a logger in our system which would has attribute log ID. Since the same logger is used by every user we have to make sure that it has only one instance at a point of time.

Thus we are using singleton design pattern to enable concurrent access.

This pattern is quite controversial since there is no flexibility due to only one instance and thus it violates open-close principle

Also it controls it's own life cycle

But In case of logger since the data flow is in one direction even if we disabled or enable logger the system isn't affected

And since all the employees would be using the same logger the use of singleton is justified

**What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?**

Designing the system really gives one a major head start in terms of delivering an effective project seeing as how one considers multiple solutions to a problem before choosing an option. Development time is now reduced

because implementation requires writing code for features who’s requirements are crystal clear, the class and the sequence diagram really help with the low level details of how methods are called and what relationships exist between various classes.

Having said all of the above, it is important that the design is verified and the best way to do that is to write some small code that validates those aspects of design which are not full proof. It may so happen that development may get halted due to flaws in the final stages of the design which means things need major revamps with too little time with a lot of undoing and redoing.

To explain it further throughout the course we were focused on developing maintainable code and thus we always followed the SOLID principles, learn the importance of refactoring and coding using interfaces to leave space for extension. We also learned about how we can make database management easy using hibernate APIs and learned about framework like spring use of plugins like tomcat maven.

It was overall good experience not in learning different languages but also developing yourself as a good programmer who uses good practices