WeWorkFiveTimesAsHard:

Project Management Tool

System Design Document

Group 5

DePaul University

Spring 2020

Contents

Document Overview

This document outlines the system design of the project management tool, WeWorkFiveTimesAsHard, being developed by Group 5.

Rationale:
The current rise of remote workers calls for an improved platform for team collaboration. A project management tool that allows participants to task progress, provide peer-feedback, and communicate with one another all on one platform would allow for efficient remote teamwork.

Document Scope:
The scope of this document is relevant to the most recent version of the project build as of 5th May 2020. We expect the project to be compatible with the most recent version of Windows 10, macOS, and the tool postgreSQL as of the date listed.

Related Documentation
https://drive.google.com/open?id=1agolWSyEjuPZno5l1COV_KDdBdREukXR

Document Conventions:
Unified Modeling Language (UML) is used throughout the document to model all architectural views.

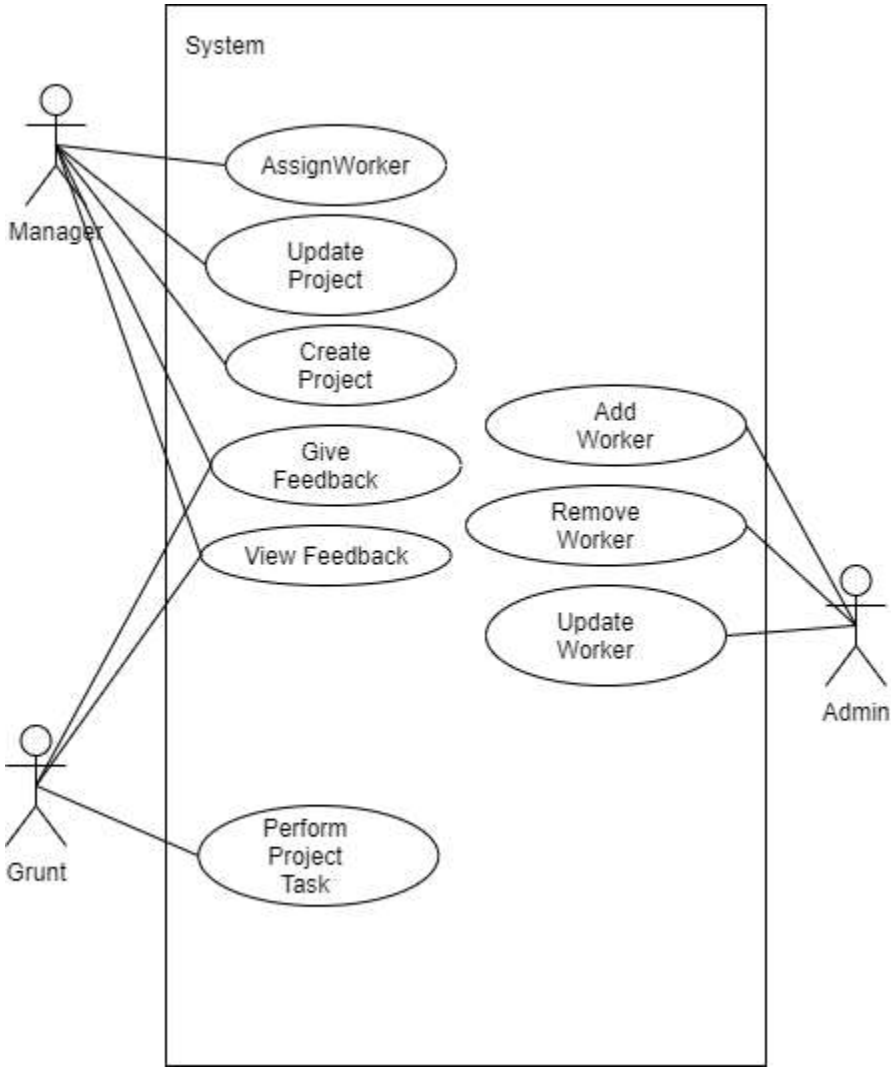
Revisions

4/5/2020	Project Outline Made
4/12/2020	Outline modified, UML Being Developed
4/13/2020	Repository setup
4/19/2020	UML Changed
4/26/2020	SQL Servers Setup
4/30/2020	Database Relations Updated
5/04/2020	Requirements Updated,

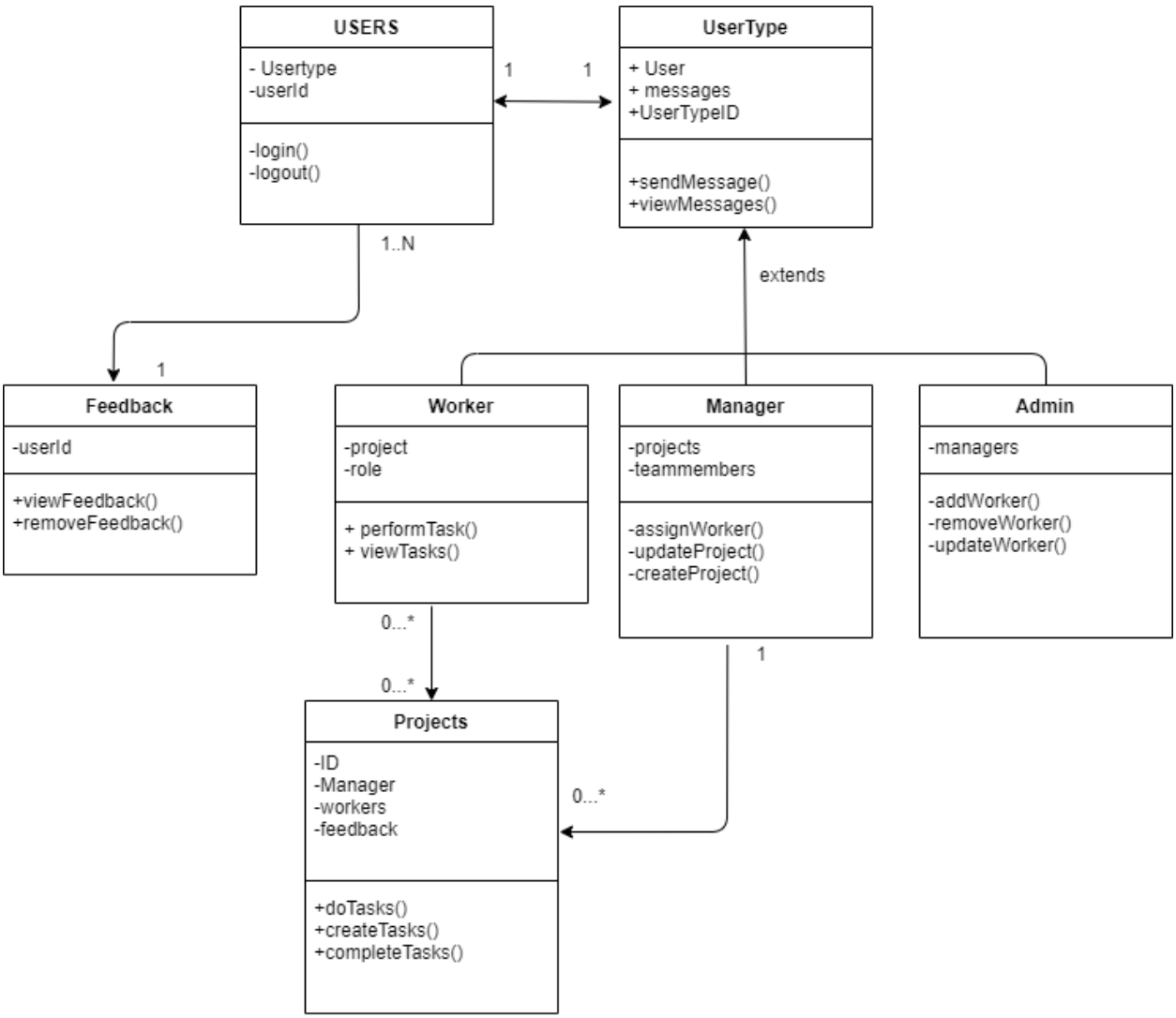
System Overview
The project management tool will allow for team members to clearly define and track the tasks needed for project completion, communicate with one another during their work, and provide feedback to one another and managers all in one centralized place.

1. System Architecture:

1.1 Low Level Use Case:



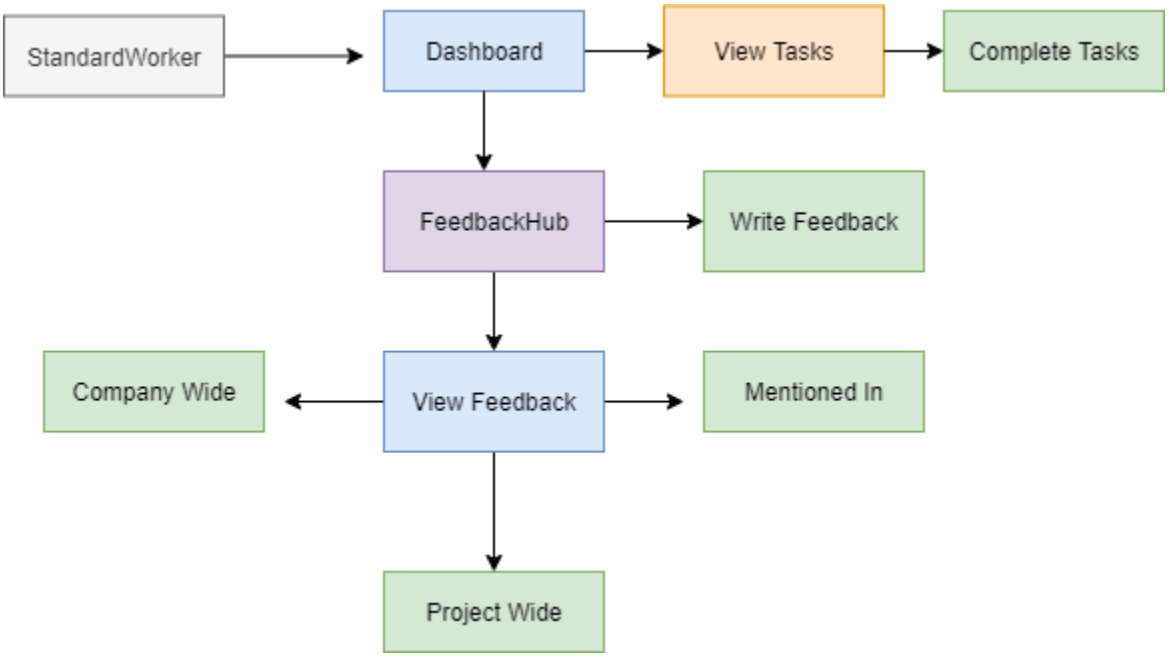
1.2 UML Diagram:



2. Detailed Use Cases:

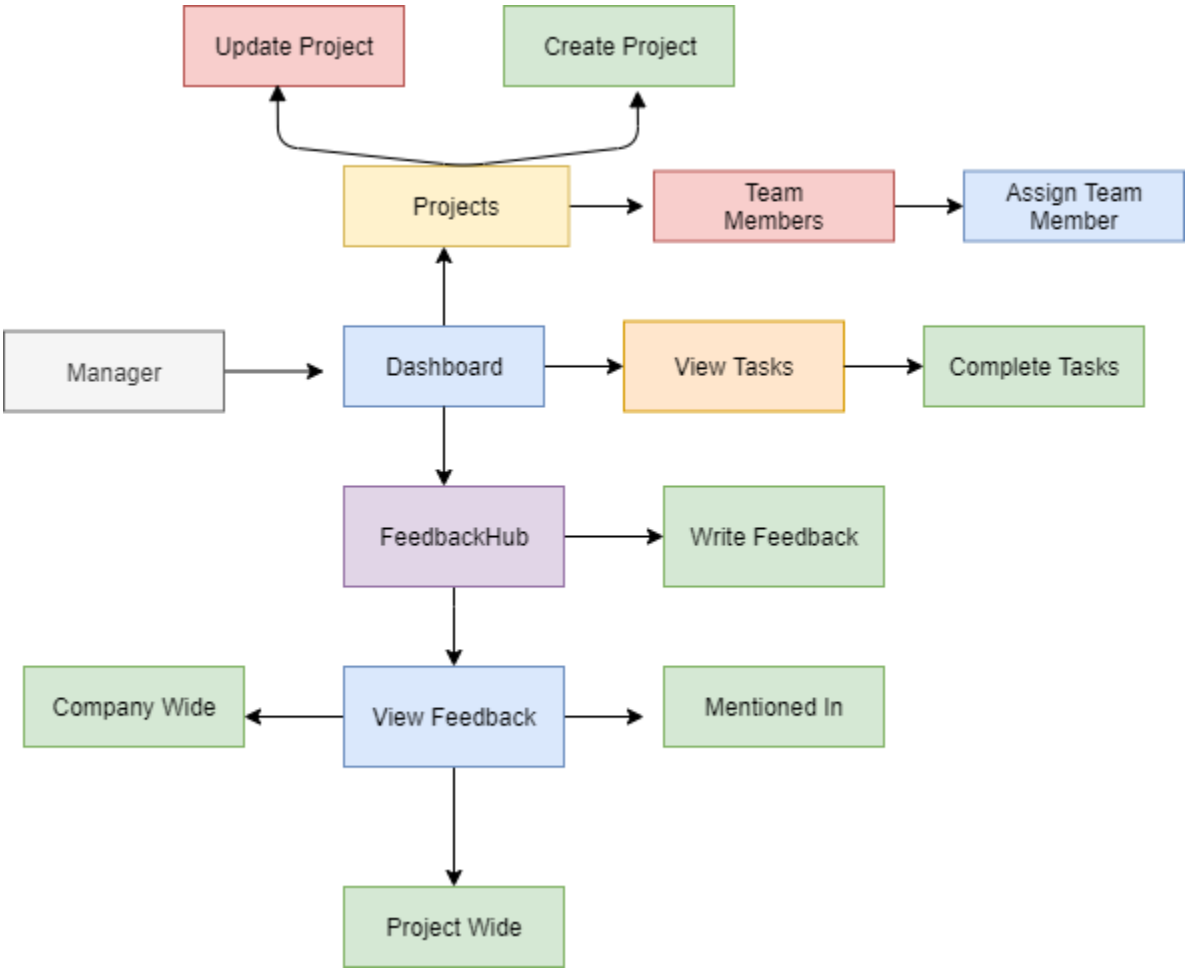
Interaction with the software will differ depending on how the user logs in, that is the *role that* a user is assigned to, changes what they are able to do. There are three types of users: Standard worker, manager, and admin. After logging these are the uses that a user has access to depending on their role.

2.1 Standard Worker:



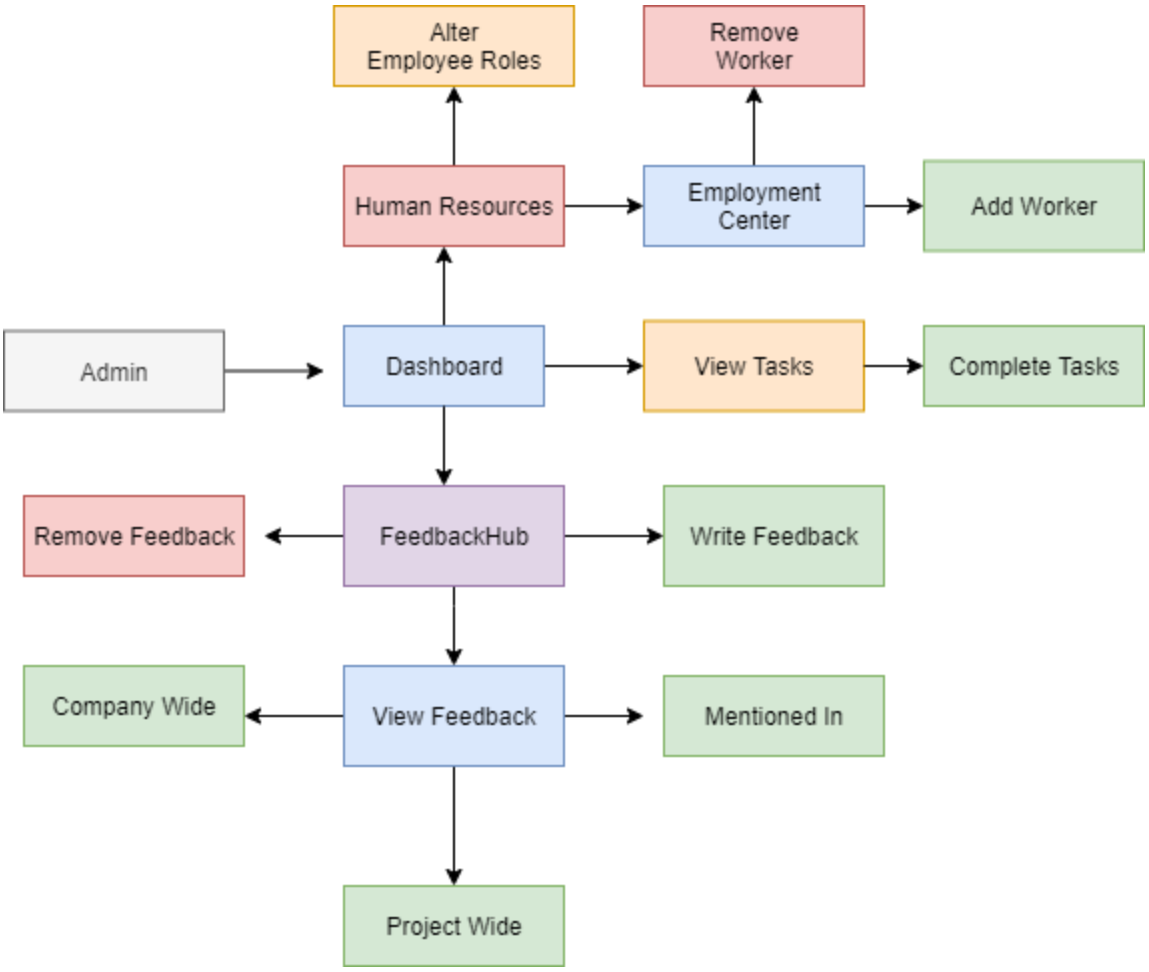
Note: A standard worker, is a team member assigned to a project, and has a superior (aka manager). A standard worker can view and complete assigned tasks, and leave feedback on a person or project. They can also view all feedback left on a project, any feedback they were mentioned in, as well as the entire company-wide feedback page.

2.2 Manager:



Note: A manager is someone who can make and update projects, assign workers to projects and tasks, as well as leave feedback on a person or project. They can also view feedback left on them, the project, or the entire company. They do not have the ability to add or remove a standard worker from their team, which we felt was best left up to the administrator; but they can assign a worker to a different project. In short, they are assigned team members and from that team, they decide who will be working on which task for a given project under their control.

2.3 Administrator:

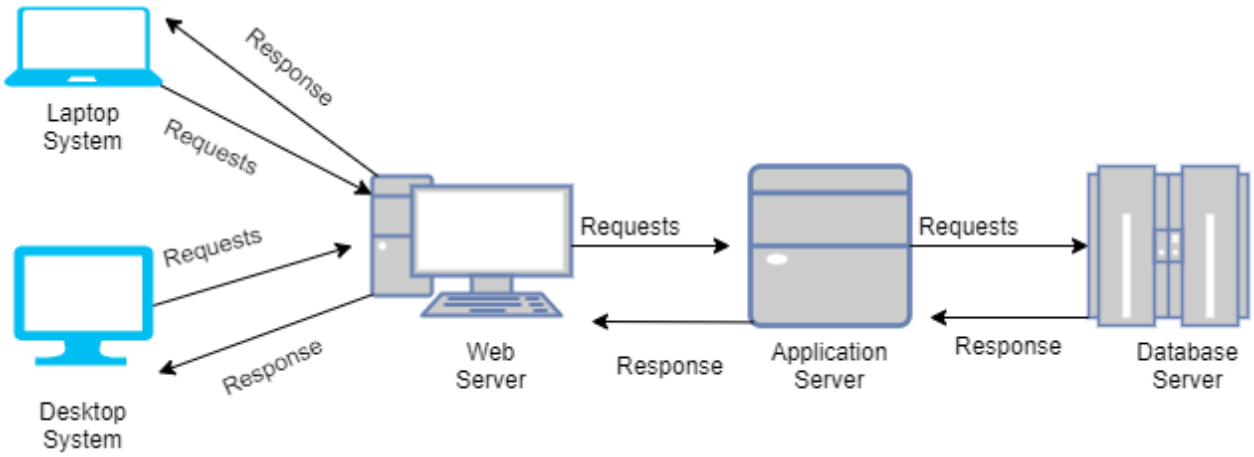


Note: An administrator can add or remove a worker from a manager and alter the role of an employee (manager or standard worker). In addition to the standard access to the feedback that a manager and a standard work can write or view, an administrator can also remove feedback left by any employee.

3. Hardware Considerations:

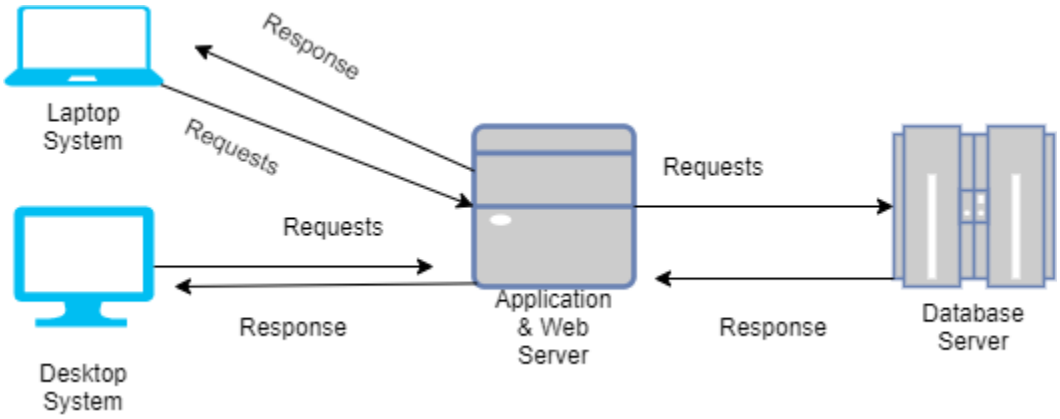
The project management tool in development will function as a web application. That is, a user goes to a webpage, logs in, and views what they have access to. What this implies, is that a user does not need to download the application, but just needs access to a client (e.g. Chrome) and can use the application at will, within the realm of their assigned role.

3.1 Preferred Hardware Setup :



In this setup, a user would have access to their client using either a Windows or macOS based desktop, or laptop system, upon which they would be served static content (the HTML & CSS) from the web server. Through interacting with the UI , the web server would then turn the request from the user, into an application server request (e.g. the user clicks a UI button), and after working out what the user intends to do, the application server can request information from the database server, getting a response, and then responding to the web server to display the appropriate page and information that the user can view (the web server's response to the user's client).

3.2 Hardware Restrictions:



Because of underlying budgetary restrictions, and the flexibility of a Spring, Maven, MVC build, our application can function just fine, with the application server acting as the web server as well to clients. In the event that our application really takes off, the servers (particularly the application & database) can be scaled up to either an in-house, or AWS service in only three-lines of configurations being changed (datasource, username, password). Though, if it ever got to the point of becoming famous enough to warrant such a change, much of the underlying implementation would need to be re-

worked due to the inherently slow nature of Java and especially SQL; the latter of which can oftentimes only be sped-up by "throwing" more hardware resources at it.

4. Software Tools:

The following tools will be used to help realize this design. The tools chosen offer a degree of flexibility for their relative performance speed, scalability, and time to development.

4.1 Core Languages:

Backend: Java 11, JPA(2.2)

Frontend: HTML, CSS

Database: SQL

Format: MVC, Maven

4.2 Project Dependencies:

Maven (3.6.3)- Project build tool used to conform to the 12 Factor Rule of application development. Namely, isolating dependencies, and storing key project configurations in the environment (e.g. port binding).

Spring Boot (2.2.1) - Used to launch our app as a service (servlet), and receive outside responses. Supports concurrency.

JPA (2.2) - Java Persistence API, enables us to interact with Java classes as if they were database tables.

ThymeLeaf (3.0.11) - Our view in the MVC. A templated server-side Java engine that allows us to develop the CSS, HMLT, JavaScript for the UI, and receive requests from the UI, to the controller.

PostgreSQL (12.2) - Our current RDBMS. Can be changed to any other RDBMS with relative ease, but was chosen for its ease of use.

FlywayDB (6.8.4) - Development tool: Used to track revisions to our RDBMS, and pre-populate tables for testing purposes.

5.Software Integration:

The core functionality of our application depends on a Maven, Spring MVC. That is, classes are separated in their respective packages as **M**odules, **V**iews, and **C**ontrollers. Maven allows us to further separate dependencies from the packages, and attach or remove dependencies on the go.

5.1 Core Application Configuration File:

There are two core configuration files that are related to spring & maven that are integral to the application being run properly.

Pom.XML: Configures all the dependencies of the maven build, including adding in support that the application runs as a Spring Boot Application with JPA, and Thymleaf.

Application.Properties: Configures how the Spring Boot application will interact with the backend. That is, it stores the necessary datasource url of our database server, as well as engage in the port binding of the application, and setting up the appropriate environmental variables of our build.

5.2 Packages (MVC):

Modules (JPA Classes)

User:

A user is someone who is able to log into the application. A user has-a usertype (strategy pattern), but its usertype is set to standard worker as a default.

UserType:

An abstract class that outlines the core functionalities of any employee, whether it is a standard worker, manager, or admin, we expect these employees to share some common abilities.

StandardWorker:

A *usertype*, can work on assigned projects and tasks, as well as leave feedback

Manager:

Is a *usertype*. Has a team, can assign team members to projects or tasks, as well as make or update projects, as well leave feedback.

Administrator:

Is a *usertype*. Can assign standard workers to a manager, or update employee roles, and add or remove an employee, as well as add or remove feedback.

Project:

Has a manager as well as standard workers. A manager can add or remove standard users to it, but a project cannot exist without a manager.

Tasks:

Made by either an administrator or manager. Becomes bound to a User on construction, done when a manager or administrator has signed off on it.

CompleteTasks:

Allows a user to work on their tasks and then mark the task as complete, which in turn puts their work up for review,

FeedbackHub:

Allows the user of anytype to view or add feedback. Special power is given to administrators to remove feedback.

ViewFeedback:

Allows the user of anytype to see feedback at the project, person, or company level.

MentionedIn:

Allows the user of anytype to see any feedback written that mentions them, whether it be at the company or project level.

ProjectWide:

Allows a user assigned to a project, to see the feedback by everyone on the project they are currently on.

CompanyWide:

Allows a user to see the feedback generated by everyone that was marked as available to view sitewide by the user who generated the feedback in the first place.

Controllers

WelcomeController:

Administrator login functionality

StandardWorkerController:

Enables a user that has a standard worker type to be able to interact with the buttons at their display in the view.

ManagerController:

Enables a user that has a manager usertype to be able to interact with the buttons at their display in the view.

AdministratorController:

Enables a user that has an administrator usertype to be able to interact with the buttons at their display in the view.

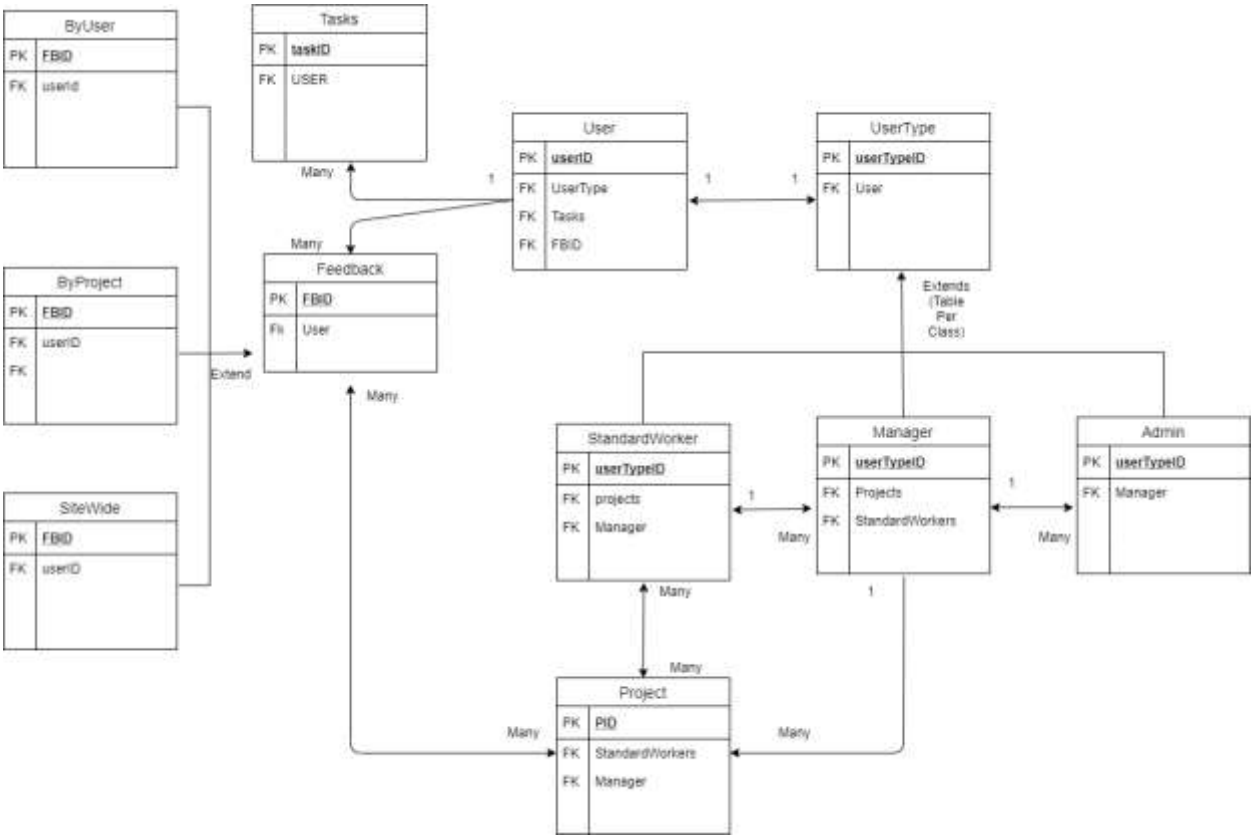
Views (UI)

All UI components will be stored in the resource package, separately as either static CSS, or HTML templates, that are compiled by tomcat through the latest version of Thymeleaf. Pages will be able to interact with the controller, which then in turn will be able interact with the modules.

6. Database Design:

The project is implemented using the Java Persistence Api. What this implies, is that we can construct Java class and then add in the @Entity dependency to them, thereby invoking JPA to take the Java class that was written, and translate that class into either an SQL table, or a NoSQL set or node. For this application, we will be using SQL, so the JPA will configure the classes into tables.

6.1 Database Layout:



Note: We will be using JPA to make the tables, that implies that the rows will be filled with any variables inside the Java-classes, which JPA will convert into SQL tables. Special attention will be paid to make the tables as optimized as possible, so that CRUD operations can be done in a relatively fast manner.

7. Performance:

Taken from Business and User Requirements

AVAILABILITY REQUIREMENTS

We aim for our product and all of its features to be accessible to users 95% of the time during a month.

RESPONSIVENESS REQUIREMENTS

We aim to make the UI response times as fast as possible:

+ Application launch < 10 seconds
+ Responses to user action < 2 seconds
+ Generating report < 30 seconds

+ Messages sent < 10 seconds

Network latency: 150 ms - 300 ms
Reference: https://github.com/Tendrl/documentation/wiki/Best-Practices-for-Response-Times-and-Latency

RELIABILITY REQUIREMENTS

We aim for our product and all of it's features to have 80 percent reliability for each month. We We will adjust this number at further stages of development.

CAPACITY REQUIREMENTS

Our system should be able to support 50 concurrent users. It will be able to manage 200 user records, and 500 project records. We will adjust this number at further stages of development.

SCALABILITY REQUIREMENTS

Our system will be able to support an annual growth of 10% for new users. We will adjust this number at further stages of development.

DISASTER RECOVERY AND BUSINESS CONTINUITY REQUIREMENTS

Since we are not working from a physical location, a disaster plan is not necessary.
To ensure business continuity in the event of problems due to server failures, widespread outages, or human error, we plan to define our Recovery Time Objective (RTO), the maximum time allowed to get things back to normal after a failure, and Recovery Point Objective (RPO), or the maximum allowable amount of lost data measured from failure to last valid backup. An example of possible numbers include:

+ ***Tier-1:*** *Mission-critical applications that require an RTPO of less than 15 minutes*
+ ***Tier-2:*** *Business-critical applications that require RTO of 2 hours and RPO of 4 hours*
+ ***Tier-3:*** *Non-critical applications that require RTO of 4 hours and RPO of 24 hours*

We will adjust these numbers at further stages of development. In order to ensure these objectives, we will have an effective backup solution for our data.

SECURITY REQUIREMENTS

USER SECURITY REQUIREMENTS

All steps necessary will be taken to ensure that user information is protected in the case of a data breach. The system will use SSL Encryption to protect all data. Strong passwords will be required for all users, and authorization/authentication schemes will be implemented to ensure safety. In addition, only users with certain permissions will be able to generate reports and

access/edit data that contains private information. A content security policy will be implemented as well, so the safety requirements and regulations are all easily accessible to all users.

DATA SECURITY REQUIREMENTS

In order to ensure data security, SSL Encryption will be used to protect all data. Data will only be able to be accessed and edited by authorized users.