4/17/2018

XSchedule Software Requirements Document

Team Members: Sonia Azad, Timothy Ratliff, Joseph Nguyen, Mark Prutz, Dylan Russell

Current Live Version: *v0.1.0*

Table of Contents

1. Preface

1.1 Summary

The following document is intended for technicians and managers to grasp a full understanding of the XSchedule application and the functions available. XSchedule is a web based application running on Amazon Web Services Elastic Beanstalk. This application will allow MODEL Computing Service customers to submit work orders online. Technicians will then easily accept these work orders and arrive on site to complete requests. With the increasing number of internet users, XSchedule will allow MODEL Computing Services to cater to a more customers. This document will inform readers of the technical specifications, features, website functionality, and other traits that XSchedule has to offer.

1.2 Version History

Current live version: *v0.1.0*
Version history: XSchedule *v0.0.1, v0.0.2*
Real requirements document inevitably contains inconsistencies, repetition, and requirement conflicts. This document too contains mistakes from the original work. This document will be reworked and rewritten multiple times to more accurately reflect the project. We find it in our best interest to provide a complete document with minimal deficiencies. Each update of XSchedule includes interface rework and system maintenance to provide a fully running application with minimal effort for our consumers. XSchedule uses the Sematic Versioning naming convention to clearly label stable builds. For more information, see: https://semver.org/
Note: The live version of XSchedule on AWS will always reflect the latest GitHub release, available at https://github.com/TimothyRatliff/XSchedule/releases.

2. Introduction

2.1 Purpose

The purpose of this document is to present a formal documentation of our web based application XSchedule (Version 1.0). This document will highlight the functions and features, technical specifications, and web functionality, as well as the application's purpose.

2.2 Product Scope

XSchedule is a web based application scheduling service for MODEL Computing Services. The application will allow customers to request a work order in real time. Customers of MODEL receive priority based on how many services they received from MODEL in the past. Technicians working for MODEL will accept work orders, from the queue, that the customers request. Technicians have the ability to accept work orders and document hours until completion. The system will then generate a bill for the customers based on working hours and technician experience. Managers have all the abilities of a technician in addition to extra abilities. Managers can view, access, and modify data about: the queue, work orders not accepted, worker performance, and times technicians are left idle. The application will be named XSchedule referencing fast scheduling services that their customers desire.

2.3 User Interface (UI)

The user interface will be different between customers, technicians and managers. Customers will be given the option to post a work order on XSchedule and will then receive a bill when a job is completed. Technicians will be able to see the queue of all the work orders and accept them. Once an order is accepted, it will be displayed in a started table indicating the work order is "in progress." Technicians will also have work ID's and years of experience listed. Managers will have information about the queue displayed on their web page. This includes: average queue length, individual technician idle times, and time the queue is empty in the form of a bar graph. Summary reports for each month will be displayed as well.

3. User Requirements Definition

3.1 Services

1. *Users.* There will be three types of users on XSchedule: Customers, Technicians, and Managers. Users will be authenticated after login to determine which category they belong to. Each user will have different interactions with XSchedule allowing features that are predetermined.
2. *Customers.* Those who use the application to request work orders. Customers will have the following information stored in their profile:
    - ID number
    - Name
    - Jobs Started

    Customers can create a work request with the following information:
    - Name
    - Complexity Level (1-3)
    - Location
    - ID number
3. *Employees.* Employees can see a list/queue of work requests. Work requests have the following information:
    - Customer priority
    - Complexity
    - Time stamp

    Employees can set a work request as active and mark when a task is completed.
4. *Managers.* Managers can view all website data, including Customer and Employee profiles, the work request queue, and the time it took for an employee to complete a task.

3.2 Nonfunctional Requirements

1. Our team will use Microsoft's Visual Studio as the primary IDE for coding. The project will use an ASP.NET Web Forms template and function as a live website, running on Amazon Web Services Elastic Beanstalk.
2. As previously mentioned, XSchedule will run on AWS Elastic Beanstalk for maximum flexibility and scalability. To minimize compatibility issues, our servers will be of the following specification:
   AWS t2.micro 64bit Windows Server 2016 v1.2.0 IIS 10.0
3. The primary modeling tool used for diagrams will follow UML as predetermined in the original requirements.
4. Group meetings are logged on a google document saved in Google Drive.

5. Our team will use the communication app Discord as the primary method of tracking deadlines and meetings. As specified in the project requirements, all code will be available on GitHub, and each member will be responsible for keeping their own contributions to the project updated through frequent commits to the GitHub repository.

3.3 Product and Process Standards

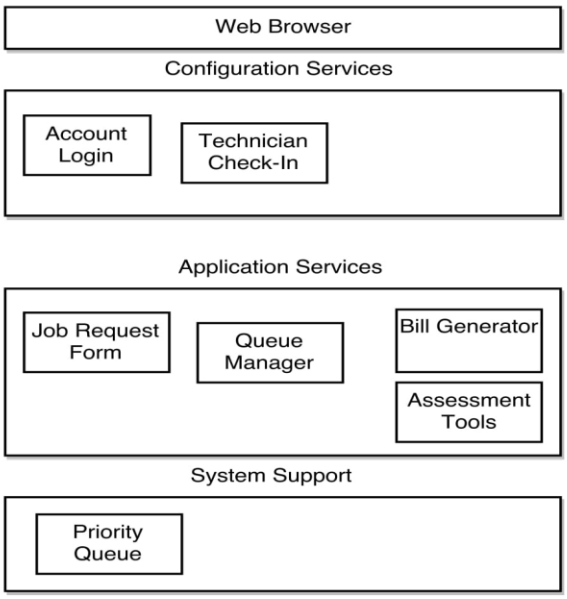
All products and processes on XSchedule will adhere to these standards:

- Safe and secure transaction process
- Requests answered with 24 - hours

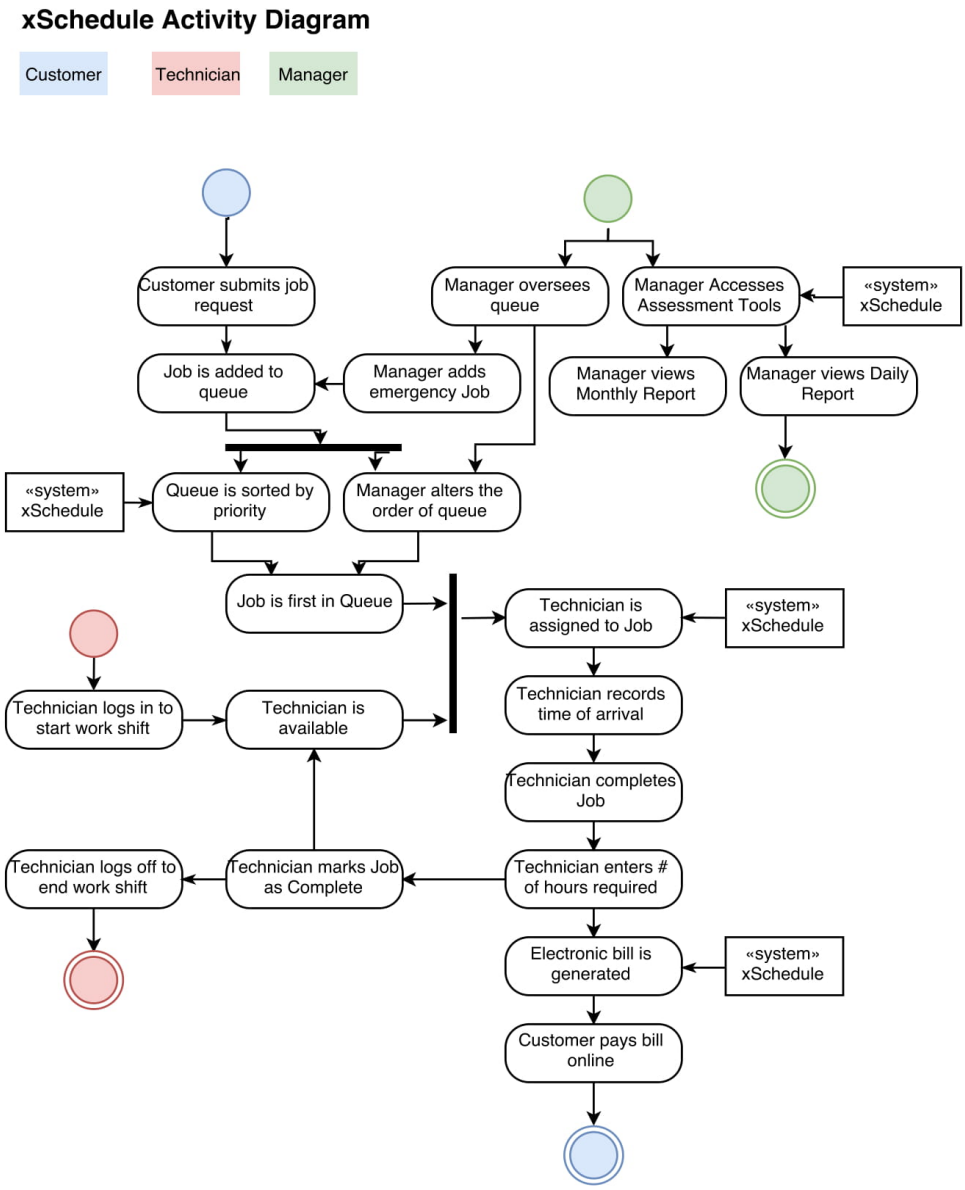
4. System Architecture

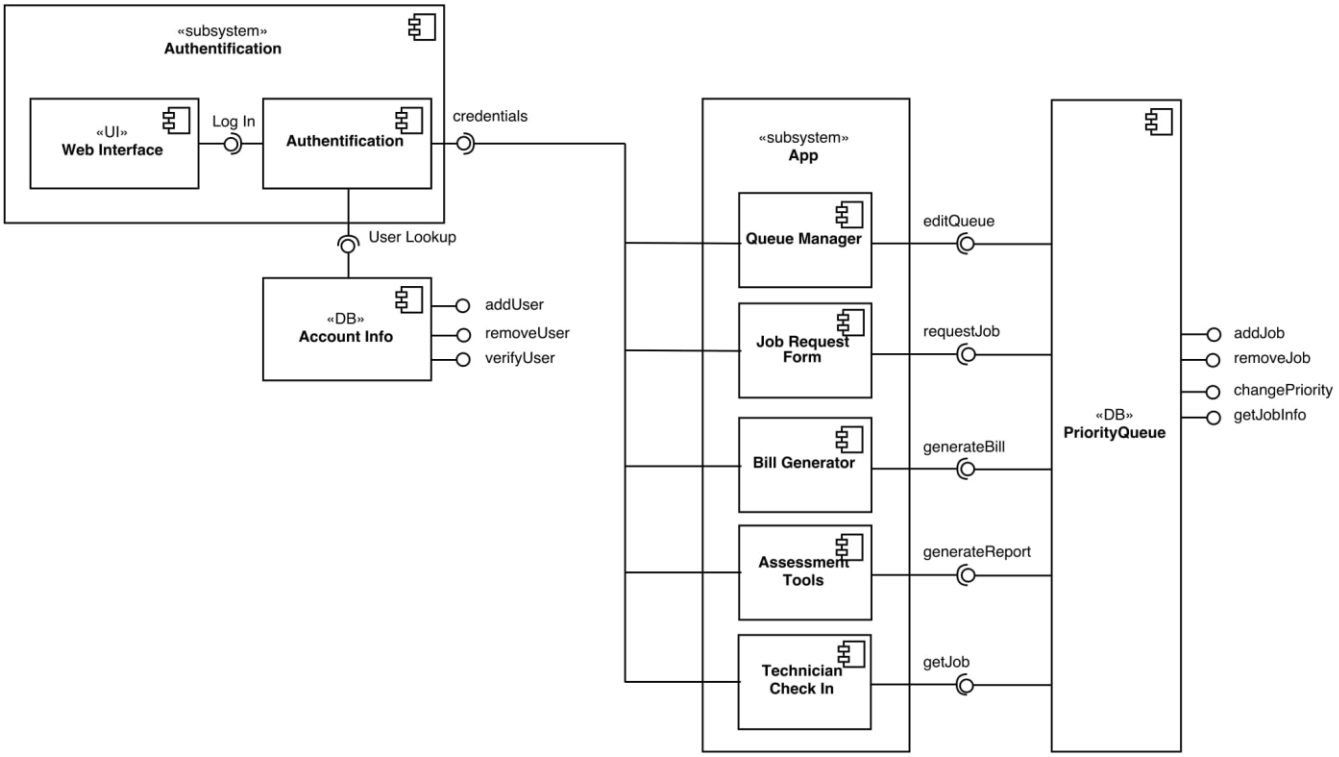
4.1 Overview

Below is the architectural model that our XSchedule system incorporates. Configuration services, application services, and system support are all incorporated when implementing XSchedule.



4.2 UML Model


xSchedule Activity Diagram
Customer
Technician
Manager
Customer submits job request
Job is added to queue
Manager adds emergency Job
Manager oversees queue
Manager Accesses Assessment Tools
«system» xSchedule
Manager views Monthly Report
Manager views Daily Report
«system» xSchedule
Queue is sorted by priority
Manager alters the order of queue
Job is first in Queue
Technician is assigned to Job
«system» xSchedule
Technician logs in to start work shift
Technician is available
Technician records time of arrival
Technician completes Job
Technician logs off to end work shift
Technician marks Job as Complete
Technician enters # of hours required
Electronic bill is generated
«system» xSchedule
Customer pays bill online



Assumptions in this document: User status (customer, technician, or employer) does not change after user creation
Job priority is set at Job Request time and does not change unless manager explicitly changes it (Ask Dr. Carver if job priority calculation should consider the customer's jobs that will be completed before the one in question or just the ones that were completed prior to requesting the job)

Note: Users will not see app functionality that should not be accessible given their credentials from authentification

4.3 Architectural Design
xSchedule Objects and Methods:

User Interface:
Fields:
+ userType: string
+ account_id: int
Methods:
+ getUserType():
return userType
+getID():
return account_id

Customer implements User:
Fields:
+ prev_jobs: Int []
+ open_jobs: Int []

Methods:
+ getPriority(): int
len = length(prev_jobs);
if(len >= 5)
return 0
else if(len < 5 && len >=3)
return 1
else if(len == 1)
return 2
else
return 3

+ getBill(int id): int
job j = getJob(Id)
return j.total_bill;

Technician implements User:
Fields:
+ assignment: Int
+ payrate: Int

Methods:
+ getJobAssignment(): void
if assignment == -1
id = getID() ;
assignment = firstInQueue(id);
return

+ enterTimeArrival(): void
    Job j = getJob(assignment);
    j.time_arrival = Datetime.Now();
    setJob(j) ;

+ enterTimeComplete: void
    Job j = getJob(assignment);
    j.time_complete = Datetime.Now();
    timeworked = j.time_complete - j.time_arrival;
    j.total_bill = payrate * timeworked;
    setJob(j) ;
    assignment = - 1;

Manager extends Technician:
Methods:
+ modifyJobRequest(int Id)
    Job j = get Job(id)
    [make changes]
    setJob(j)
    return;

Job Object:
Fields:
+ job_id: int
+ customer_id: int
+ complexity: int
+ priority: int
+ time_submit: time

+ technician_id: int
+ time_arrival: time
+ time_complete: time
+ bill_total: int

---

queueSystem Object:
Fields:
+ users: List of <User>???
+ queue: Int []

+ ongoing: Int[]
+ alljobs: List of <Job>
+ jobcounter: int = - 1

Methods:

+ getJob(int id) : Job
    if (id <= jobcounter)
        return alljobs[id]
    else return Null

+ setJob(Job j): void
    int i = j.job_id;
    alljobs[i] = j;
    return j;

+ createJob(Customer c, Int difficulty): void
    Job j = new Job();
    jobcounter = jobcounter + 1;
    j.job_id = jobcounter;
    j.customer_id = c.getID();
    j.priority = c.getPriority();
    j.complexity = difficulty;
    j.time_submit = DateTime.Now();
    alljobs.add(j);
    [add job to “open” in order of priority]
    return;

+ firstInQueue(int id) : int
    if len(open) > 0
        int id = open[0];
        alljobs[id].technician_id = id;
        open = open.remove(0);
        ongoing = ongoing.add(Id) ;
        return id;
    else
        return -1

+ getAvgWait(): int
    Return Math.avg(wait);
+ getAvgLength(): int
    Return Math.avg(length);
+ getPrecentageEmpty(): int
    Return PercentageEmpty();
+ getIncompleteJobs(): int
    Return IncompleteJobs();
+ getTechicianHours(): List of <Technician, idleHours(int)>

```
        Return technician.idleHours();
+ generateDailyReport
        if(System.Time != OverTime)
                Return today.report();
        Else
                Return today.report() + OverTime();
+ generateMonthlyReport
        if(MonthlyReport() != OverTime())
                Return month.report();
        Else
                Return moth.report() + OverTime();
```

5. System Requirements Specifications

5.1 Functional/Nonfunctional

1. The system will be implemented as a client-server system with information help on a SQLite database.
2. Client access to the system will be accessible through a standard web browser. The primary web browser chosen for XSchedule will be Google Chrome.
3. User selections will be made using menus predetermined.
4. User inputs will be validated according to rules established by the development team when the system user interface is designed. Users will be notified of invalid inputs by the system.
5. The system will have a login feature for customers, managers, and technicians. If username match to the corresponding password in SQLite3 database, login will be successful otherwise refer to 4.
6. A real time queue will be updated every time a job is added or taken out of the queue.
7. The system will have an in-progress table for jobs in-progress that updates every time a job is taken or completed.
8. Work order estimates will be generated by the system based on imputed factors of technician experience and hours spent until job completion.
9. All customers that submit work orders will be notified by the system when their job is completed. A job is completed when a technician marks the job as complete.
10. All managerial positions maintain the right to access the queue and accept/remove work orders as well as modify the queue if one work order is in urgent need of completion.
11. Data regarding average wait time before a job is started, average queue length, percentage of time the queue is empty, number of jobs not addressed on their request date and number of hours each day a technician is idle will be recorded by the system.
12. All managerial positions maintain the right to access the data regarding the monthly reports given.

5.2 Queue Specification

The following information about the queue will be recorded as management has requested:

1. Average wait time before a job is started

a. Will be calculated, within the system, by adding each individual work order queue time and dividing it by the number of jobs in the queue for that particular day.

2. Average queue length

. This can be accomplished by finding the mean of the time each work order is in the queue.

3. Percentage of time the queue is empty

. This can be calculated by taking the amount of time the queue is empty and dividing that time by the time the queue is not empty multiplied by 100.

4. Number of jobs not addressed on their request date

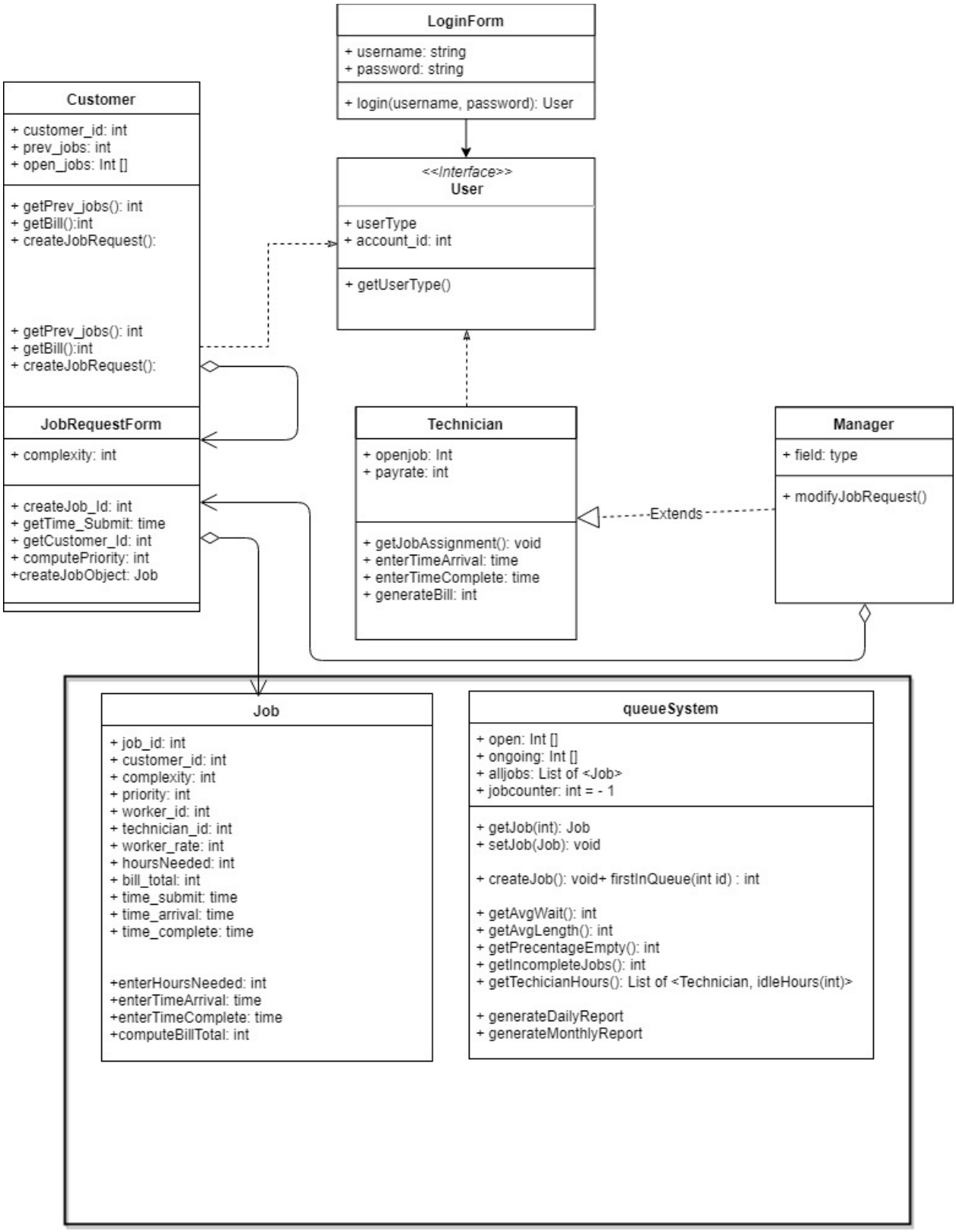
. Will be calculated with a counter.

5. Number of hours each day a technician is idle

. This will be calculated, within the system, by subtracting the average work day time (8 hours) by the number of hours the technician is clocked in.

6. System Models

6.1 UML Models


LoginForm
+ username: string
+ password: string
+ login(username, password): User
Customer
+ customer_id: int
+ prev_jobs: int
+ open_jobs: Int []
+ getPrev_jobs(): int
+ getBill():int
+ createJobRequest():
+ getPrev_jobs(): int
+ getBill():int
+ createJobRequest():
<<Interface>>
User
+ userType
+ account_id: int
+ getUserType()
JobRequestForm
+ complexity: int
+ createJob_Id: int
+ getTime_Submit: time
+ getCustomer_Id: int
+ computePriority: int
+createJobObject: Job
Technician
+ openjob: Int
+ payrate: int
+ getJobAssignment(): void
+ enterTimeArrival: time
+ enterTimeComplete: time
+ generateBill: int
Extends
Manager
+ field: type
+ modifyJobRequest()
Job
+ job_id: int
+ customer_id: int
+ complexity: int
+ priority: int
+ worker_id: int
+ technician_id: int
+ worker_rate: int
+ hoursNeeded: int
+ bill_total: int
+ time_submit: time
+ time_arrival: time
+ time_complete: time
+enterHoursNeeded: int
+enterTimeArrival: time
+enterTimeComplete: time
+computeBillTotal: int
queueSystem
+ open: Int []
+ ongoing: Int []
+ alljobs: List of <Job>
+ jobcounter: int = - 1
+ getJob(int): Job
+ setJob(Job): void
+ createJob(): void+ firstInQueue(int id) : int
+ getAvgWait(): int
+ getAvgLength(): int
+ getPrecentageEmpty(): int
+ getIncompleteJobs(): int
+ getTechicianHours(): List of <Technician, idleHours(int)>
+ generateDailyReport
+ generateMonthlyReport

7. System Evolution

7.1 Overview

Login

All users must login with a username and password (stored in a secure database) to access the application. The login page will check the credentials entered, and if deemed correct, grant access to the application.

Register

In order to access XSchedule, the user must register a username and password on the register page. The register page will accept these credentials and save them to the secure database.

About

The about page will contain information about the project, purpose of the application, and the course (Software System Development).

Contact

The contact page will contain links to each group member's GitHub profile page, and a short bio about the member.

8. Appendices

8.1 Hardware Requirements

The host server for XSchedule must be able to compile and run an ASP.NET Web Forms app. The minimum hardware requirements to do this are as follows:

- 1 Gigabyte of RAM
- Minimum of 1.4 GHZ x64 processor

8.2 Database Requirements

SQLite3 database required to store data from XSchedule's system. Data includes information regarding the queue, technician work times, and user accounts.

9. Group Information

9.1 Meetings

		Sonia Azad (Team Leader)	Dylan Russell	Joseph Nguyen	Mark Prutz	Timothy Ratliff	Notes:
2/9/2018 1:00pm-2:30pm	Meeting #1	1.5	1.5	1.5	1	1.5	Mark could only come from 1:30pm-2:30pm
2/21/2018 5:00pm-6:30pm	Meeting #2	1.5	0	1.5	1.5	1.5	Dylan couldn't attend
3/8/2018 12pm-2pm	Meeting #3	1.5	2	2	0	0	Tim had class
3/14/18 9pm - 10pm	Meeting #4	1	1	1	1	1	Discord becomes main method of meeting communications
3/19/2018 6pm-7pm	Meeting #5	1	1	1	1	1	Discord: discussion of Architectural Documents
4/4/2018 3:00pm-5:00pm	Meeting #6	2	2	2	2	2	Mark made a prototype and presented it to us.
4/11/2018 10pm - 11:30pm	Meeting #7	1.5	1.5	1.5	1.5	1.5	Voice Chat on Discord
4/14/2018 1pm-3pm	Meeting #8	1	2	2	2	0	Tim had band event
	Total Hours:	11	11	12.5	10	8.5	
					Group Total:	53	(Group meeting hours, not total)

9.2 Contributions

XSchedule Project Contributions

Use Cases	Dylan Russell and Mark Prutz
Requirements Document	Joseph Nguyen and Timothy Ratliff, Revisions by Joseph
Activity Diagram	Sonia Azad
Architectural Diagram	Sonia Azad
Component Diagram	Mark Prutz
Class Diagram & Descriptions	Sonia Azad
Test Cases	Dylan Russell
PowerPoint Presentation	Joseph Nguyen Revisions by Team H
GitHub Setup & Wiki	Timothy Ratliff
Prototype	Mark Prutz
Home Page UI	Joseph Nguyen

For GitHub Contributions, see
https://github.com/TimothyRatliff/XSchedule/graphs/contributors