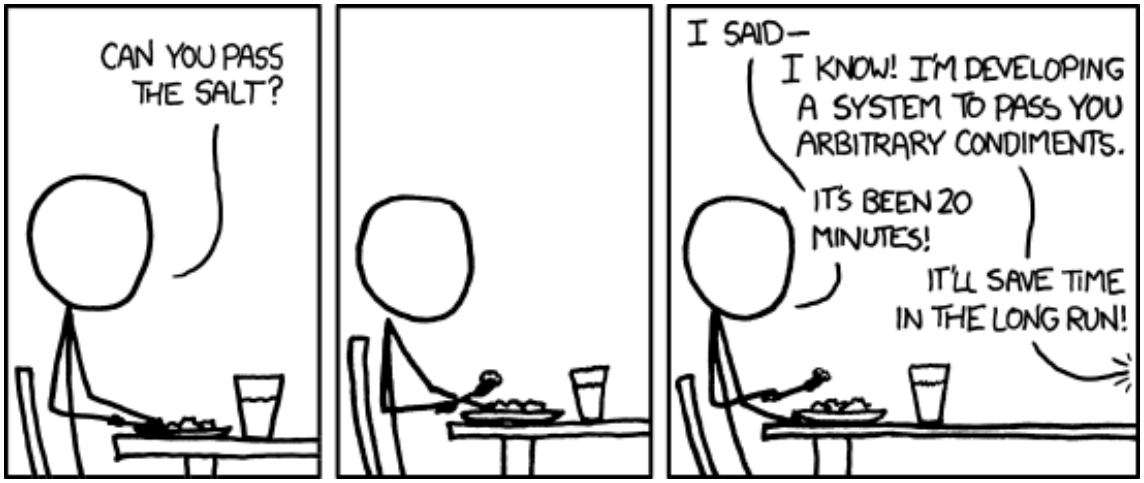
Swimming Pool Automated Checking System

CITS4401 Software Requirements and Design - Practical Assignment

Mitchell Pomery (21130887)

April 30, 2015



source: xkcd 974

Contents

1 Introduction

The Swimming Pool Automated Checking System (SPACS) helps to keep track of and assist in the upkeep of private swimming pools. This document outlines the design of the SPACS system and is intended to be used as a reference guide by anyone involved in the creation the SPACS system.

1.1 Terms

Below are a list of terms and abbreviations used in this document and their definitions.

API	Application Programming Interface - A set of functions that allow the manipulation of the system through defined procedures.
PTU	Pool Testing Unit - Takes readings from a pool and sends it to the system
SPACS	Swimming Pool Automated Checking System

2 Use Case Diagram

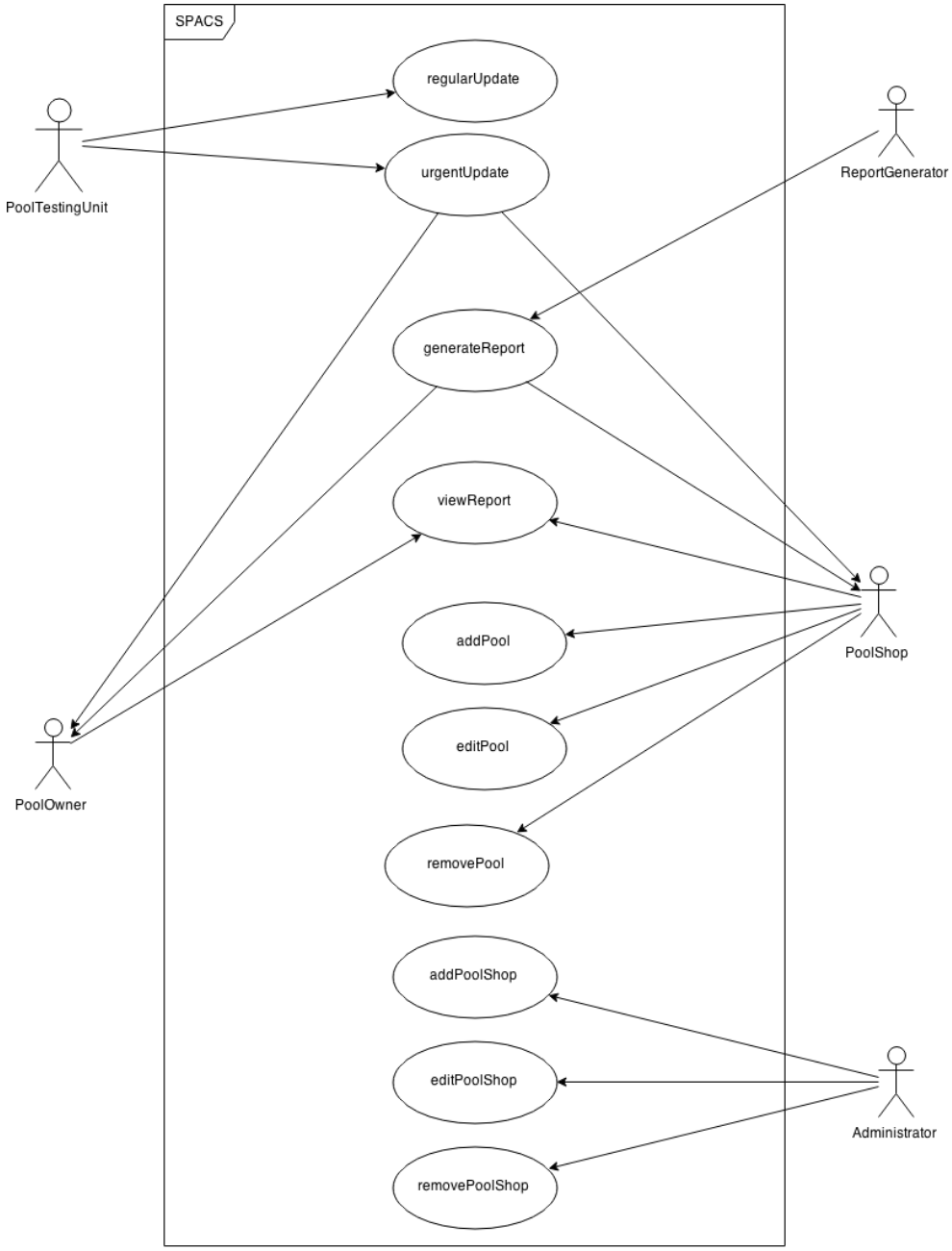


Figure 1: use case diagram outlining the main uses for the system

Name	regular_update
Actors	PoolTestingUnit
Goal	Store information received from the PoolTestingUnit in the system.
Preconditions	PoolTestingUnit is authenticated
Basic Flow	1. Use case starts when PTU sends data. 2. Data is checked for any errors. 3. Records from the data are stored for later analysis. 4. Use case ends.
Alternative Flow	1. The data does not validate (a) The data is logged for analysis by support.
Postconditions	1. Success: Data has been stored. 2. Failure: Data has been stored in a log for analysis by an Administrator.

Name	urgent_update
Actors	PoolTestingUnit, PoolShop, PoolOwner
Goal	Store collected information received from the PoolTestingUnit in the system and alert the PoolOwner and PoolShop that there is a problem.
Preconditions	PoolTestingUnit is authenticated
Basic Flow	1. Use case starts when PoolTestingUnit sends data with alerts. 2. Data is checked for any errors. 3. Records from the data are stored for later analysis. 4. An email is sent to the PoolShopOwner and PoolOwner. 5. Use case ends.
Alternative Flow	1. The data does not validate (a) Received data is logged for analysis (b) Email has been sent to PoolShopOwner and PoolOwner
Postconditions	1. Success: Data has been stored and an email has been sent to PoolShopOwner and PoolOwner. 2. Failure: Data has been stored in a log for analysis by an Administrator and an email has been sent to PoolShopOwner and PoolOwner.

Name	generate_report
Actors	ReportGenerator, PoolOwner, PoolShop
Goal	Provide latest information about a pool to the PoolOwner and the PoolShop
Preconditions	First week of the PTU or a month since the last report for each pool
Basic Flow	1. Use case starts at the same time every day. 2. Gets a list of pools that need reports generated. 3. For report that needs generating: (a) Gets the information that should be on the report. (b) Generates the report as a pdf. (c) Emails the report off. (d) Saves a copy of the report for future reference.
Alternative Flow	1. Error generating report (a) Failure is logged for analysis by Administrator.
Postconditions	1. Success: Report generated and emailed to pool owner and pool shop 2. Failure: Any errors logged for admin to look over

Name	add_pool_shop
Actors	Administrator, PoolShop
Goal	To add a pool shop to the system.
Preconditions	Administrator is authenticated.
Basic Flow	1. Use case starts when an Administrator goes to the add pool shop page. 2. Administrator user enters information about the pool shop. 3. Information is validated to check for errors. 4. Information is stored and a new PoolShop is created.
Alternative Flow	1. Invalid Information (a) Administrator user is informed (b) Administrator can modify information and try again.
Postconditions	1. Success: Information is stored and a new PoolShop is created. 2. Failure: No change to the system.

Name	edit_pool_shop
Actors	Administrator, PoolShop
Goal	To edit a pool shop in the system.
Preconditions	Administrator is authenticated.
Basic Flow	1. Use case starts when an Administrator goes to the edit pool shop page. 2. Administrator user changes information about the pool shop. 3. Information is validated to check for errors. 4. Information is stored and a PoolShop is updated.
Alternative Flow	1. Invalid Information (a) Administrator user is informed (b) Administrator can modify information and try again.
Postconditions	1. Success: Information is stored and PoolShop is updated. 2. Failure: No change to the system.

Name	remove_pool_shop
Actors	Administrator, PoolShop
Goal	To remove a pool shop from the system.
Preconditions	Administrator is authenticated.
Basic Flow	1. Use case starts when an Administrator goes to the remove pool shop page. 2. Administrator confirms they want to remove PoolShop.
Alternative Flow	1. Action Canceled (a) No change to the system.
Postconditions	1. Success: PoolShop is no longer usable. 2. Failure: No change to the system.

Name	add_pool
Actors	PoolShop, PoolTestingUnit
Goal	To add a pool to the system.
Preconditions	PoolShop is authenticated.
Basic Flow	1. Use case starts when a PoolShop goes to the add pool page. 2. PoolShop user enters information about the pool. 3. Information is validated to check for errors. 4. Information is stored and a new PoolTestingUnit is created for the pool.
Alternative Flow	1. Invalid Information (a) PoolShop user is informed (b) PoolShop can modify information and try again.
Postconditions	1. Success: Information is stored and a new PoolTestingUnit is created. 2. Failure: No change to the system.

Name	edit_pool
Actors	PoolShop, PoolTestingUnit
Goal	To edit a pool in the system.
Preconditions	PoolShop is authenticated.
Basic Flow	1. Use case starts when a PoolShop goes to the edit pool page. 2. PoolShop user changes information about the pool shop. 3. Information is validated to check for errors. 4. Information is stored and a PoolTestingUnit is updated.
Alternative Flow	1. Invalid Information (a) PoolShop user is informed (b) PoolShop can modify information and try again.
Postconditions	1. Success: Information is stored and PoolTestingUnit is updated. 2. Failure: No change to the system.

Name	remove_pool
Actors	PoolShop, PoolTestingUnit
Goal	To remove a pool from the system.
Preconditions	PoolShop is authenticated.
Basic Flow	1. Use case starts when and PoolShop goes to the remove pool page. 2. PoolShop confirms they want to remove PoolTestingUnit.
Alternative Flow	1. Action Canceled (a) No change to the system.
Postconditions	1. Success: PoolTestingUnit is no longer usable. 2. Failure: No change to the system.

3 Object Models

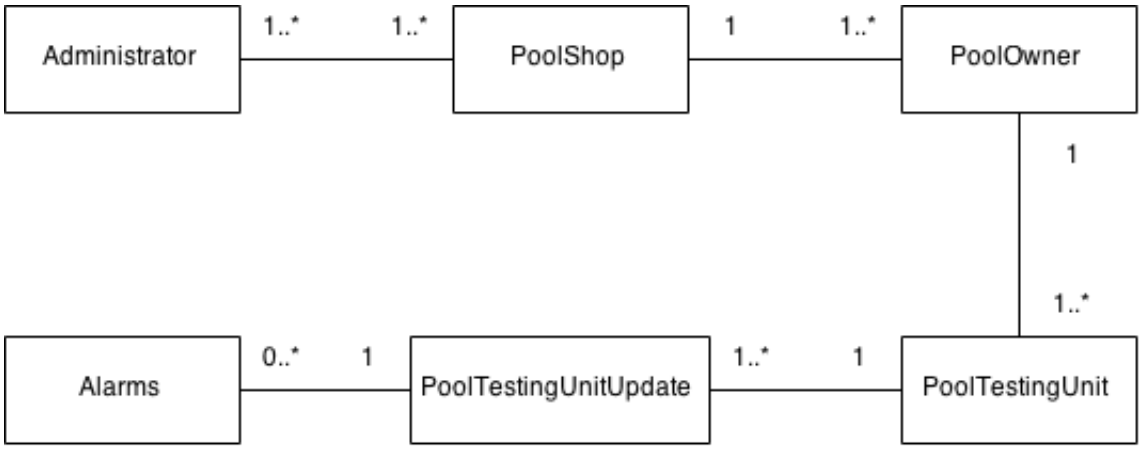


Figure 2: relationships between the main classes in the system

There can be multiple administrators in the SPACS system that manage all the PoolShopAdmin. Each PoolOwner can only be registered at one PoolShopAdmin, and each PoolTestingUnit can only be linked to one PoolOwner. The Administrator, PoolShopAdmin and PoolOwner objects all contain information about a single person.

The above figure shows the basics of how the different objects interact, and not the helper classes that ensure these links are done correctly. More information on how these classes interact can be found below.

3.1 Objects

User

User is the class that all users of the system will fall under. It ensures a minimum amount of information is collected about each user and will implement all the basic instructions needed for interacting with the objects. The level property allows the user object to do this. Authentication information is stored in the Authentication object to minimize the ability to leak sensitive information, such as passwords from it.

User(object)
+ id + title + name + address + email_address + phone_number + mobile_number + level
+

Authentication(object)
+ id + username + password
+

Administrator, PoolShop and PoolOwner

These are all extensions of the User object and will implement any extra methods that they do not inherit. Pythonic code doesn't use setter and getter functions, but still allows the properties to have validation on them.

Administrator(User)
+
+

PoolShop(User)
+ pool_owners
+

PoolOwner(User)
+ pool_testing_units
+

ShopOwnerLink

ShopOwnerLink stores the link between a PoolShopAdmin and a PoolOwner, ensuring that a PoolShopAdmin can only see pools that they manage.

ShopOwnerLink(object)
+ pool_shop_admin_id + pool_owner_id
+

PoolTestingUnit

Pool Testing Units are mostly passive users of the system and have the sole role of providing information to the system. They are closely linked to their PoolOwner. Again, due to the way python works this object should have no methods.

PoolTestingUnit(object)
+ ptu_id + owner_id + length + width + depth + volume + above_ground + material
+

PoolTestingUnitUpdate

PoolTestingUnitUpdate objects store all the information from the records that the pool testing units send.

PoolTestingUnitUpdate(object)
+ report_id + ptu_id + datetime + pH + chlorine_level + chlorinator_status + total_alkalinity + temperature + water_hardness + water_level + datetime_last_filter + alarms
+

Alarm

Alarm objects store alarms set off when a PoolTestingUnit sends a PoolTestingUnitUpdate.

PoolTestingUnitUpdate(object)
+ alarm_id + report_id + reason + value + expected
+

4 Sequence Diagrams

4.1 regular_update

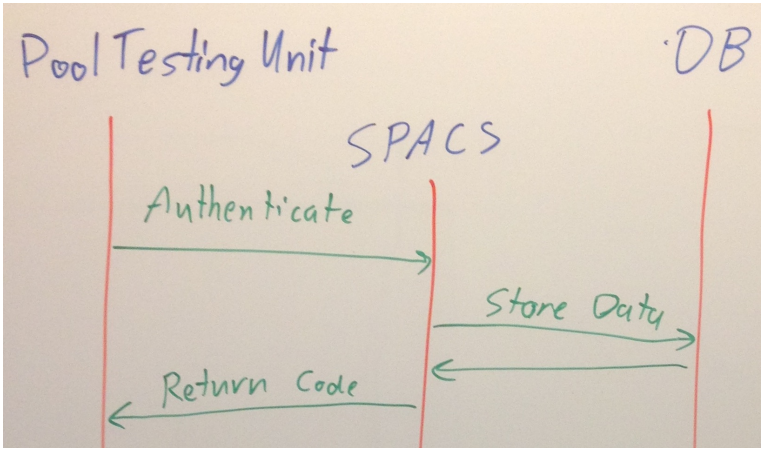


Figure 3:

4.2 urgent_update

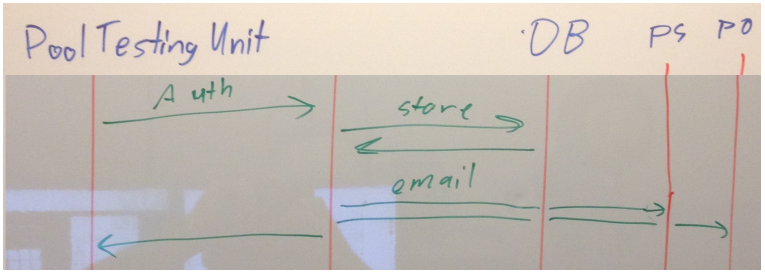


Figure 4:

4.3 generate_report

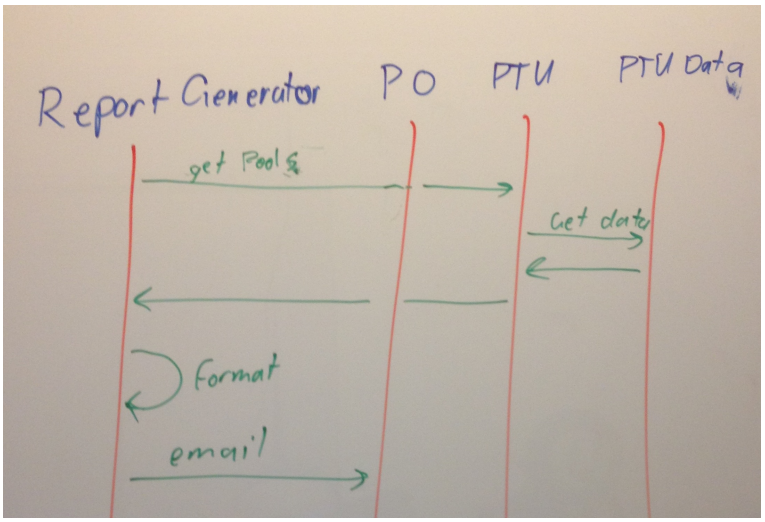


Figure 5:

4.4 edit_pool_shop

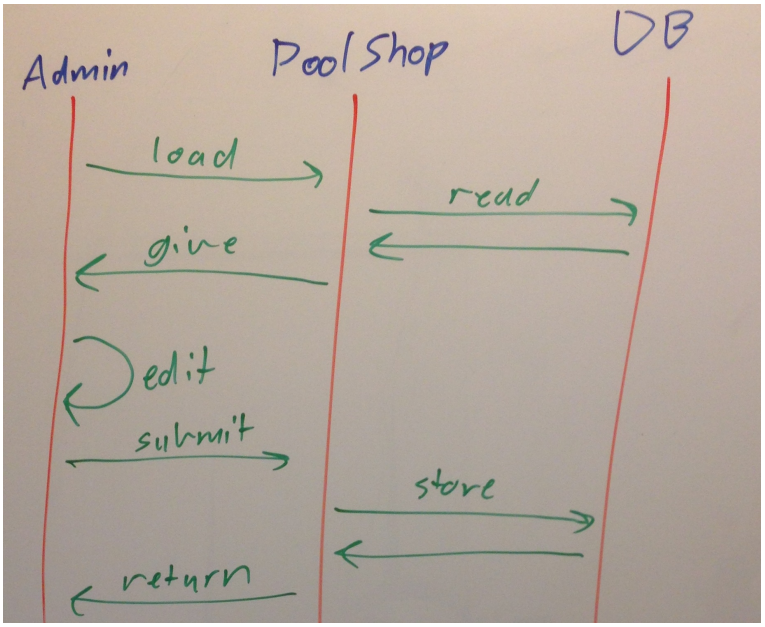


Figure 6:

4.5 edit_pool

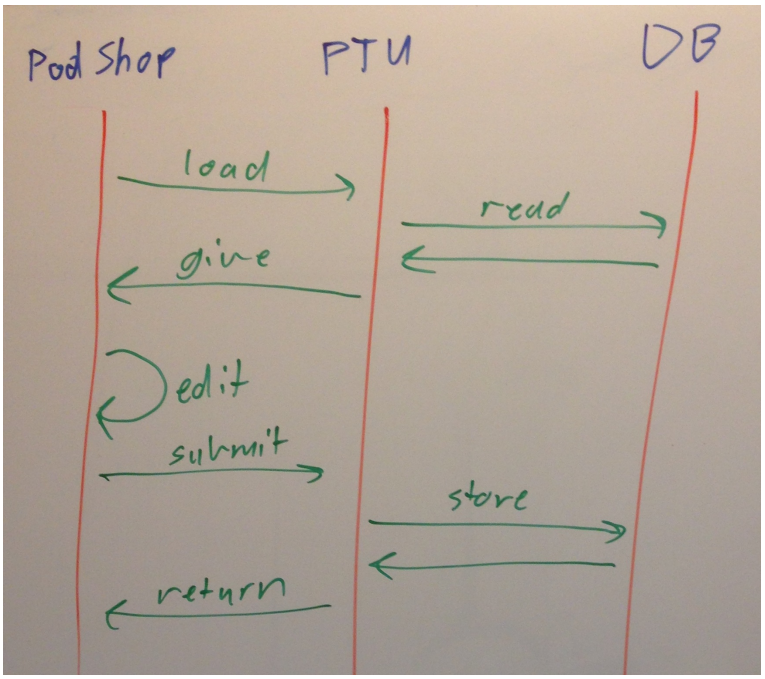


Figure 7:

5 Design Considerations

Decisions have been made to meet the quality requirements of the SPACS system listed below. These decisions aim to maximize each requirement while keeping the cost of development and running the system as low as possible.

1. Performance
2. Reliability
3. Usability
4. Portability
5. Modifiability
6. Future Requirements

5.1 Python

Python has been chosen as the implementation language over other object oriented languages such as Java due to it's lightweight footprint and portability. Python allows for rapid development and testing, making the system easy to modify. Well written Python code also has the advantage of being self documenting, meaning that is highly readable so less comments are required.

Unlike other object oriented languages, Python is dynamically typed. This means that the types of each variable are not defined while developing, but rather at run time. Python is also an interpreted language, rather than a compiled one. This means that things such as type errors and non existent variables will only be found during run time. A full set of testing and good logging will minimize the risk of this causing issues while the system is in production.

As python has been chosen, class and method names in this document are named according to the Python PEP 8 Style Guide.

5.2 Statelessness

The application has been developed with scalability in mind meaning that all sessions should be stateless and information should not be managed by the program once it has finished with it. It will be possible to set up multiple instances of the server that are able to communicate with the same database. This will allow the application to scale in the event where the number of users increases past an individual servers load.

Adding this scalability will require a load balancer to sit between the users and the server instances. The addition of the load balancer will also allow individual servers to be taken offline for updates or if there is an issue with a machine.

5.3 API Based

Keeping the relationships between all the objects are kept as simple as possible minimizes the need for complex helper classes. As such, everything can be implemented as API calls made straight from the web application. This leads itself to making the application scalable, as as much of the work as possible is rendered at the client side.

5.4 Transaction Beans

Transaction Beans are found in many enterprise Java applications and make modifying data in the database simple. They allow the object to be loaded from a database bean and will store any changes made to the object when completed. All object loading and saving will be done in the transaction beans, and the structure of in the code will mirror that of the database. As transaction beans centralize all database access any issues with the database and all logging of database requests will be dealt with here.

5.5 Reliability

The SPACS system has been designed with reliability in mind. The ability to run multiple SPACS servers in a cluster minimizes the chance of downtime. It is also self contained and will restart itself after any fatal errors. Logging and email alerts also mean that an Administrator monitoring the system can easily find and diagnose problems.

6 Subsystems

The SPACS system will be broken down into several subsystems. These subsystems ensure that the work is done in a logical manner and allow for the system to easily be expanded on in the future. Interaction between the subsystems should be minimal as they all have highly defined roles.

1. Server
   (a) Website
   (b) API
   (c) Scheduler
2. Database

6.1 Server

The server subsystem is responsible for running the main portion of the program. It will start up all the subsystems below it according to a global configuration file. Any errors that the systems below it cause will be caught by the server and handled gracefully. It will also be responsible

for making sure that any information from the systems below it are logged correctly.

Website

The website will be the main user interface and will be managed by the server. All connections to this will be stateless, meaning that several servers can be launched behind a load balancer and act together so that the system can be scaled up as the number of users increases if needed.

API

The API will run on top of the Website and will be the only way that a user can interface with the database. This ensures that all the features the end user sees exist in one place. All Object manipulation will be managed calls implemented in the API.

Scheduler

The scheduler will be responsible for running anything that is timing sensitive, such as report generation, or that may need to be retried, such as emailing. This ensures that all retry and timing logic will appear in only one location. Any thing that needs to be scheduled will be stored such that it can be accessed after the server has been restarted. Items that are scheduled will be responsible for setting their own timings and retry logic allowing the flexibility for them to react differently depending on their own outcomes.

6.2 Database

The database will be the one true source of all information for the system. Objects will be loaded out of it and stored back in it when they are finished with. Starting and keeping it running will not be a role of server subsystem.

7 State Charts

7.1 Server

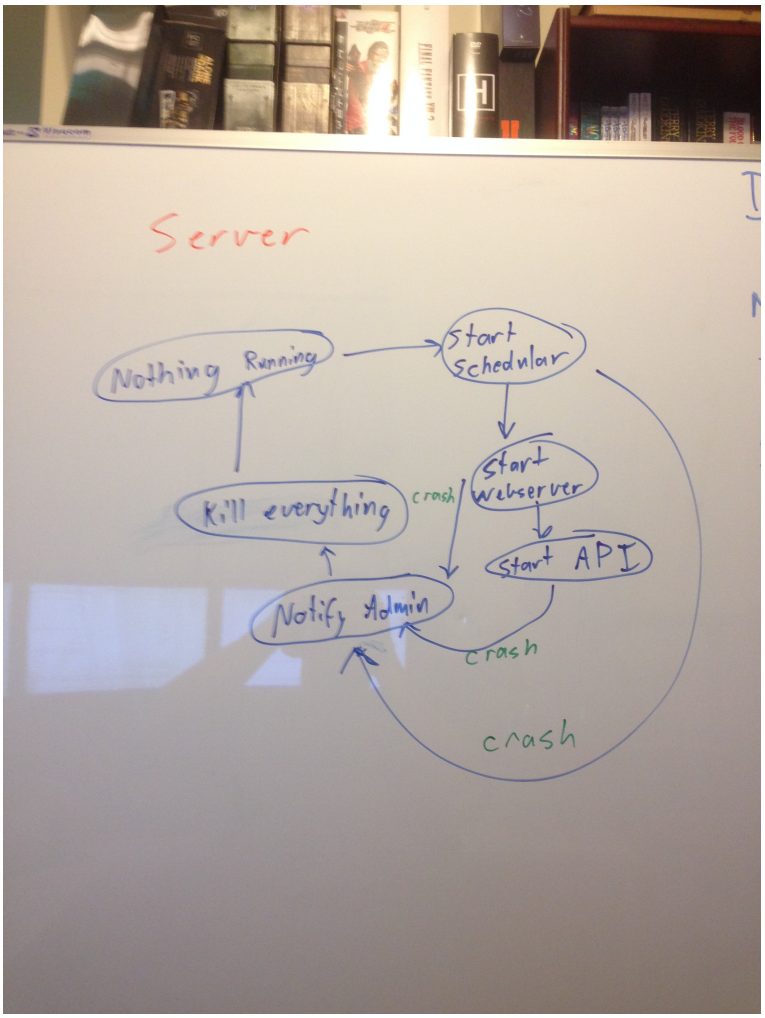


Figure 8:

7.2 Scheduler

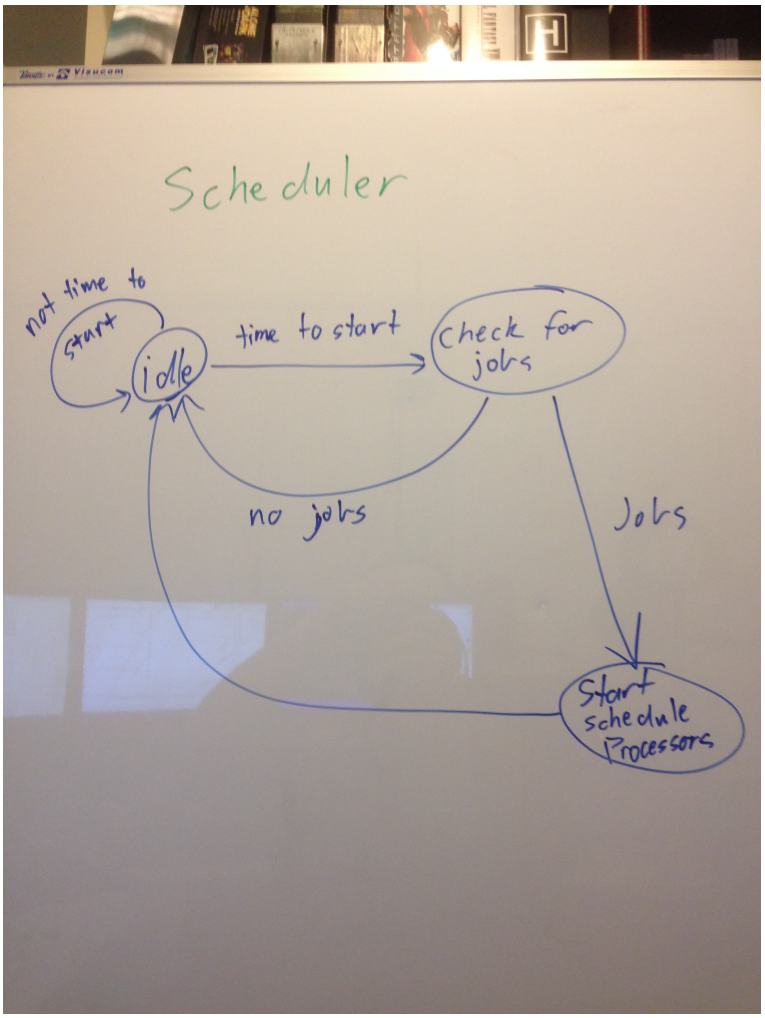


Figure 9:

States:

- Idle
- Checking For Jobs
-

8 Design Pattern

CITS4401 Software Requirements and Design - Practical Assignment. CITS4401 Software Requirements and Design - Practical Assignment. CITS4401 Software Requirements and Design - Practical Assignment. CITS4401 Software Requirements and Design - Practical Assignment. CITS4401 Software Requirements and Design - Practical Assignment. CITS4401 Software Requirements and Design - Practical Assignment.