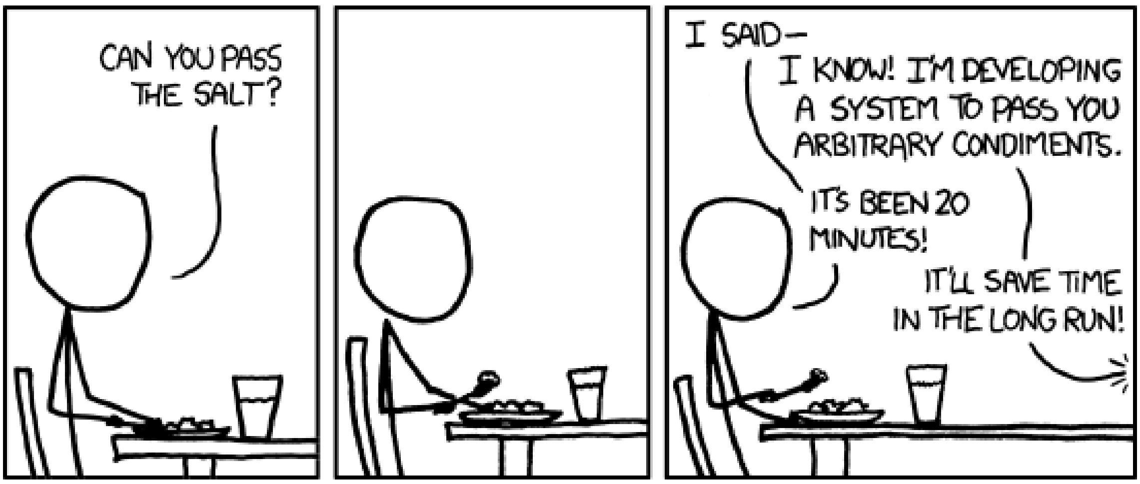
# Swimming Pool Automated Checking System

CITS4401 Software Requirements and Design - Practical Assignment

Mitchell Pomery (21130887)

April 25, 2015



# 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 |
| SPACS | Swimming Pool Automated Checking System |

# 2 Use Case Diagram

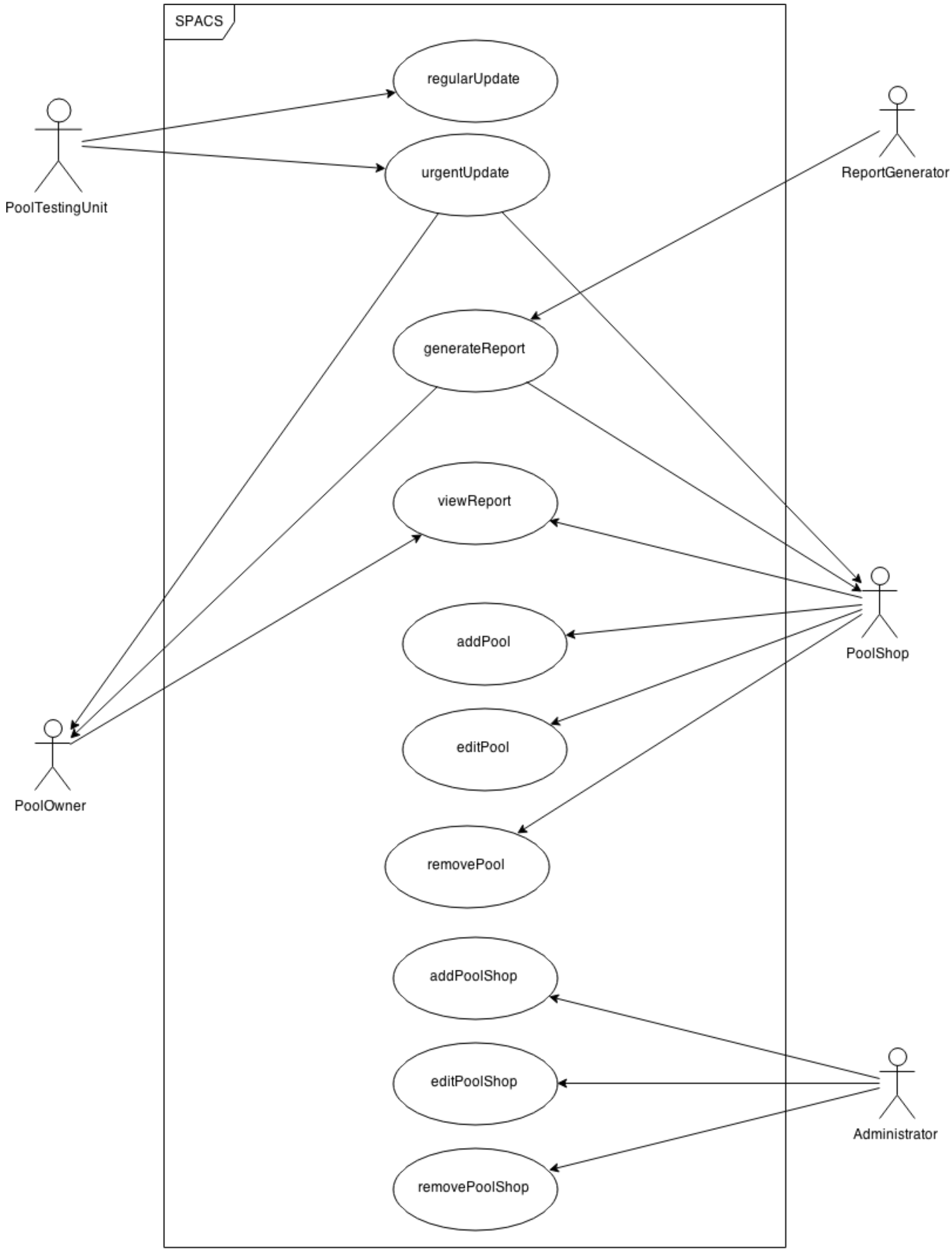



Figure 1: use case diagram outlining the main uses for the system

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

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

| | |
|---|---|
| **Name** | generate_report |
| **Actors** | ReportGenerator, PoolOwner, PoolShopAdministrator |
| **Goal** | Provide latest data to the |
| **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.<br>2. Gets a list of pools that need reports generated.<br>3. For report that needs generating:<br>(a) Gets the information that should be on the report.<br>(b) Generates the report as a pdf.<br>(c) Emails the report off.<br>(d) Saves a copy of the report for future reference. |
| **Alternative Flow** | 1. Error generating report<br>(a) Failure is logged for analysis by Administrator. |
| **Postconditions** | 1. Success: Report generated and emailed to pool owner and pool shop<br>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.<br>2. Administrator user enters information about the pool shop.<br>3. Information is validated to check for errors.<br>4. Information is stored and a new PoolShop is created. |
| **Alternative Flow** | 1. Invalid Information<br>(a) Administrator user is informed<br>(b) Administrator can modify information and try again. |
| **Postconditions** | 1. Success: Information is stored and a new PoolShop is created.<br>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.<br>2. Administrator user changes information about the pool shop.<br>3. Information is validated to check for errors.<br>4. Information is stored and a PoolShop is updated. |
| **Alternative Flow** | 1. Invalid Information<br>(a) Administrator user is informed<br>(b) Administrator can modify information and try again. |
| **Postconditions** | 1. Success: Information is stored and PoolShop is updated.<br>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.<br>2. Administrator confirms they want to remove PoolShop. |
| **Alternative Flow** | 1. Action Canceled<br>(a) No change to the system. |
| **Postconditions** | 1. Success: PoolShop is no longer usable.<br>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.<br>2. PoolShop user enters information about the pool.<br>3. Information is validated to check for errors.<br>4. Information is stored and a new PoolTestingUnit is created for the pool. |
| **Alternative Flow** | 1. Invalid Information<br>(a) PoolShop user is informed<br>(b) PoolShop can modify information and try again. |
| **Postconditions** | 1. Success: Information is stored and a new PoolTestingUnit is created.<br>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.<br>2. PoolShop user changes information about the pool shop.<br>3. Information is validated to check for errors.<br>4. Information is stored and a PoolTestingUnit is updated. |
| **Alternative Flow** | 1. Invalid Information<br>(a) PoolShop user is informed<br>(b) PoolShop can modify information and try again. |
| **Postconditions** | 1. Success: Information is stored and PoolTestingUnit is updated.<br>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.<br>2. PoolShop confirms they want to remove PoolTestingUnit. |
| **Alternative Flow** | 1. Action Canceled<br>(a) No change to the system. |
| **Postconditions** | 1. Success: PoolTestingUnit is no longer usable.<br>2. Failure: No change to the system. |

# 3 Object Models

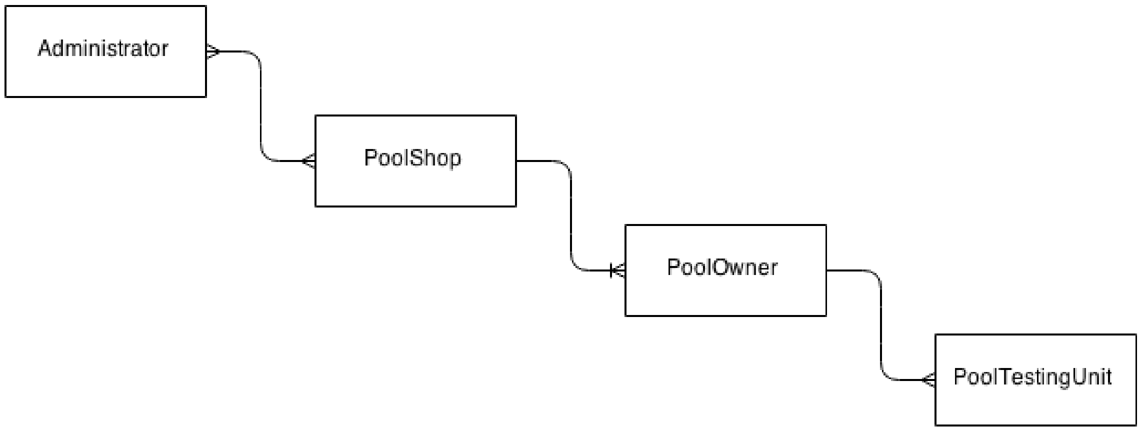



Figure 2: relationships between the main classes in the system

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

## 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.

**Administrator, PoolShop and PoolOwner**

These are all extensions of the User object and will implement any extra methods that they do not inherit.

**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.

# 4 Sequence Diagrams

# 5 Design Considerations

The following are a list of considerations that have been made. These choices affect what the system will look like.

1. Python
2. Statelessness
3. Simple Relationships
4. Transaction Beans

## 5.1 Python

Python has been chosen as the implementation language over other object oriented languages such as Java. This system is small and python allows for rapid development. Well written Python code also has the advantage of being self documenting, meaning less time has to be spent writing comments and keeping them up to date.

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 be able to be passed from any server to any other server. Multiple instances of the server should also be able to communicate with the 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 Simple Relationships

The relationships between all the objects are kept as simple as possible to minimize the need for complex helper classes.

## 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 by a transaction bean and will store any changes made to the object once it is no longer needed. This means that all object loading will be done in the one location, and the structure of in the code will mirror that of the database.

Transaction beans centralize all database access. This means that any issues are dealt with in this one location, as well as the logging of any requests.

# 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.

#### 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. It will not be managed by the server subsystem.

# 7 State Charts

## 7.1 Server

## 7.2 Scheduler

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.

# 9 Dynamic Models

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.