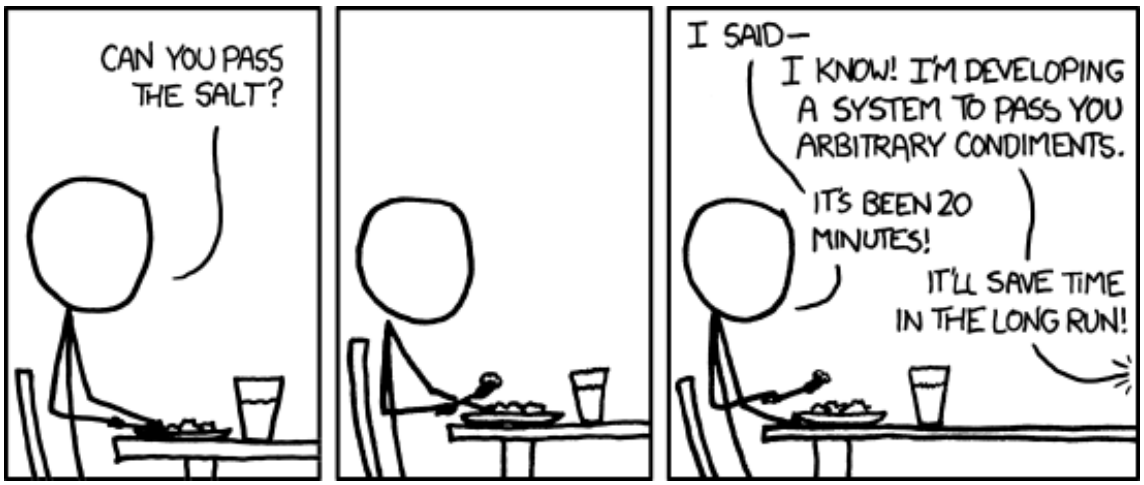
Swimming Pool Automated Checking System

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 to ensure that it meets all the requirements. It can also be used as a reference guide by anyone developing 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

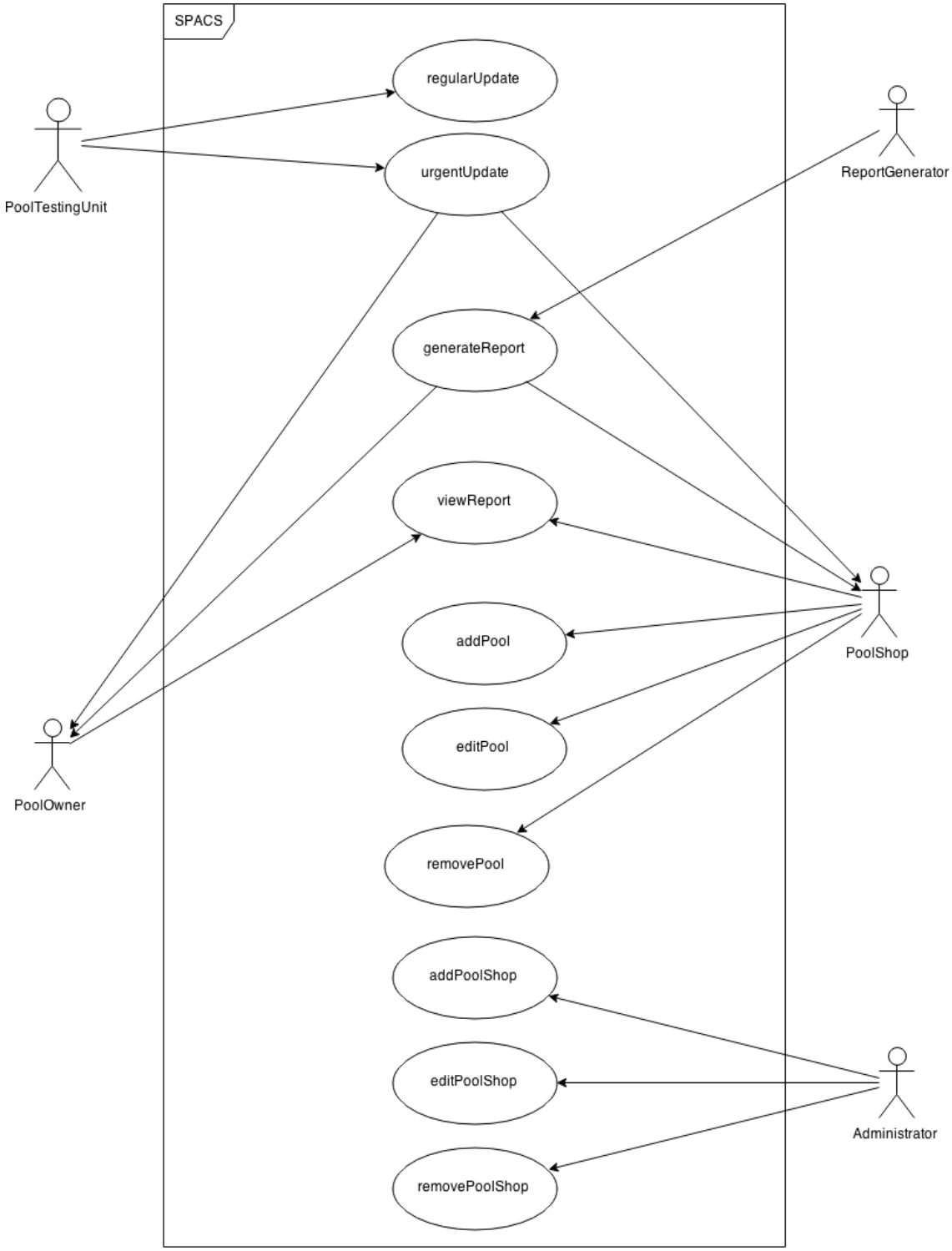


Figure 1: use case diagram outlining the main uses for the system

Name	regularUpdate
Actors	PoolTestingUnit
Goal	store collected information from the PTU in the system
Preconditions	PoolTestingUnit is authenticated
Basic Flow	1. Use case starts when ptu sends data 2. validate data 3. store it so that it can be used later 4. The use case ends
Alternative Flow	1. Data is malformed (a) Recieved data is logged for analysis 2. Issue storing data (a) Fall back to logging data and alert the administrator
Postconditions	1. Success: Data has been stored 2. Failure: Data has been stored in a log for analysis by admin

Name	urgentUpdate
Actors	PoolTestingUnit, PoolShopAdministrator, PoolOwner
Goal	store collected information from the PTU in the system and alert the pool owner and pool shop that there is a problem
Preconditions	PoolTestingUnit is authenticated
Basic Flow	1. use case starts when ptu sends data with alerts 2. validate data 3. store it so that it can be used later 4. email is sent to the PoolShopOwner and PoolOwner 5. the use case ends
Alternative Flow	1. Data is malformed (a) Recieved data is logged for analysis 2. Issue storing data (a) Fall back to logging data and alert the administrator 3. Email fails (a) Email gets retried and event is logged
Postconditions	1. Success: Data has been stored, email has been sent to PoolShopOwner and PoolOwner 2. Failure: Data has been stored in a log for analysis by admin

Name	generateReport
Actors	PoolOwner, PoolShopAdministrator
Goal	provide latest data to
Preconditions	First week of the PTU or a month since the last report
Basic Flow	1. use case starts at the same time every day 2. gets a list of pools that need reports 3. for each pool 4. gets the information that should be on the report 5. generates the report as a pdf 6. emails it off
Alternative Flow	
Postconditions	1. Success: Report generated and emailed to pool owner and pool shop 2. Failure: Any errors logged for admin to look over

Name	addPoolShop
Actors	Administrator
Goal	To add a pool shop to the system.
Preconditions	
Basic Flow	1. Administrative user enters information about the pool shop
Alternative Flow	- Invalid Information - Error displayed and user is able to re-enter
Postconditions	Success: Data is stored and can be retieved later Failure: User is given achance to modify data

Name	editPoolShop
Actors	Administrator
Goal	To edit a pool shop in the system.
Preconditions	
Basic Flow	1. Administrative user enters updated information about the pool shop
Alternative Flow	1. Invalid Information (a) Error displayed and user is able to re-enter
Postconditions	1. Success: Data is stored and can be retieved later 2. Failure: User is given achance to modify data

Name	removePoolShop
Actors	Administrator
Goal	To remove a pool shop from the system.
Preconditions	
Basic Flow	1. Administrative selects the pool shop 2. Confirms that the pool shop should be disabled
Alternative Flow	1. Cancelled (a) No change is made
Postconditions	1. Success: Data is no longer accessible. User no longer able to log in 2. Failure: No change

Name	addPool
Actors	PoolShopAdministrator
Goal	To add a pool to the system.
Preconditions	
Basic Flow	
Alternative Flow	
Postconditions	

Name	editPool
Actors	PoolShopAdministrator
Goal	To edit a pool in the system.
Preconditions	
Basic Flow	
Alternative Flow	
Postconditions	

Name	removePool
Actors	PoolShopAdministrator
Goal	To remove a pool from the system.
Preconditions	
Basic Flow	
Alternative Flow	
Postconditions	

3 Object Models

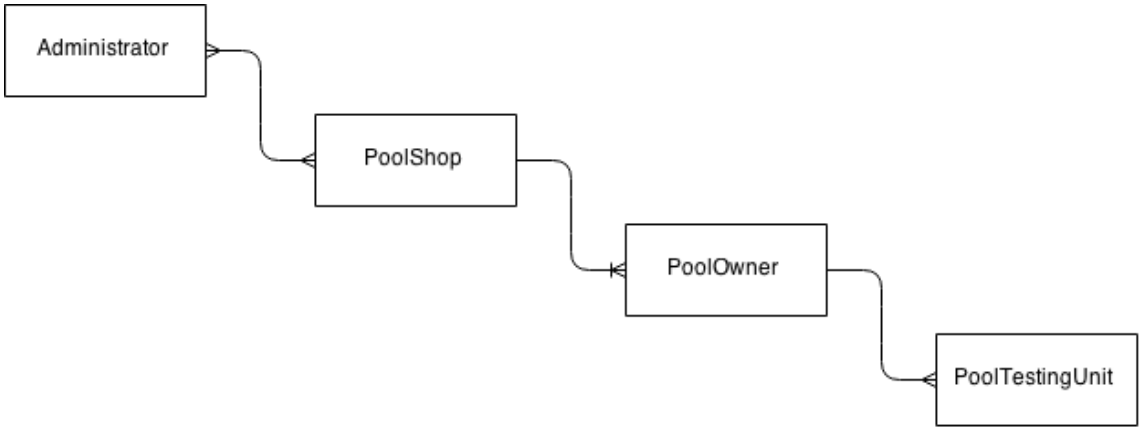


Figure 2: relationships between the main classes in the system

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.

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.

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.