

# **BeeSecured Web – An AJAX Web Interface to a Sensor Network for Occupational Safety**

**by**

**Frank Chen**

B.A.Sc., Simon Fraser University, 2008

Research Project Submitted in Partial Fulfillment  
of the Requirements for the Degree of  
Master of Engineering

in the

School of Engineering Science  
Faculty of Applied Sciences

**© Frank Chen 2012**

**SIMON FRASER UNIVERSITY**

**Spring 2012**

All rights reserved.

However, in accordance with the *Copyright Act of Canada*, this work may be reproduced, without authorization, under the conditions for "Fair Dealing." Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.

# Approval

**Name:** Frank Chen  
**Degree:** Master of Engineering  
**Title of Thesis:** *BeeSecured Web – An AJAX Web Interface to a Sensor Network for Occupational Safety*

**Examining Committee:**

**Chair:** Ash Parameswaran, Professor

---

**Bozena Kaminska**  
Senior Supervisor  
Professor

---

**Kamal Gupta**  
Supervisor  
Professor

**Date Defended/Approved:** January 25, 2012

---

## Partial Copyright Licence



The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the "Institutional Repository" link of the SFU Library website ([www.lib.sfu.ca](http://www.lib.sfu.ca)) at <http://summit/sfu.ca> and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library  
Burnaby, British Columbia, Canada

revised Fall 2011

## Abstract

A software application for monitoring and controlling sensor network data needs to be easily accessible and needs to maintain an accurate record of all activities in the network. Traditional desktop applications have been a primary form of software interface for many applications, but they require additional setup per physical machine without the flexibility of access from additional computers. Our particular system of interest, the BeeSecured rapid deployment and monitoring system, an end-to-end solution for remotely monitoring personnel and assets in various terrains using wireless sensor networks (WSN), currently uses a desktop application as the primary software interface and does not include capabilities to maintain a record of the network activities.

This report presents the design and implementation of an AJAX application with the integration of an open-source map technology and database system. The purpose of this project is to design a cross-browser compatible user interface that can be conveniently accessed by any computing device with an internet connection. The application is suitable for real-time information updates as well as securely controlling any system level data. In particular, the application is targeted for displaying geographical locations for sensors in a WSN, with the capability to store and retrieve the most up-to-date sensor network data in real-time. We have implemented an application that utilizes AJAX technology at run-time, with the integration of the Google Maps version 2 technology and the PostgreSQL database system, and we demonstrated our application by integrating it with the BeeSecured rapid deployment and monitoring system. WSN devices were generated via a software simulator to test the various functionalities of our application, and we were able to demonstrate a working prototype of the web interface.

**Keywords:** BeeSecured Web; BeeSecured; Google Maps; database; sensor network; GWT

## **Acknowledgements**

First, I would like to thank Dr. Bozena Kaminska for giving me the opportunity to partake in the BeeSecured project and her guidance through various stages of my M.Eng degree. I would also like to thank Dr. Marcin Marzencki for coordinating project scheduling and providing general support throughout the project, and Dr. Jens Wawerla for providing software and technical support, documentation, setting up required environments for application deployment and driving the software integration effort during the development of the project. I would also like to extend my thanks to all other BeeSecured members who were involved in various stages in the project and provided additional support and project knowledge.

# Table of Contents

Approval.....	ii
Partial Copyright Licence .....	iii
Abstract.....	iv
Acknowledgements .....	v
Table of Contents.....	vi
List of Tables.....	viii
List of Figures.....	x
List of Acronyms.....	xii
<b>1. Introduction .....</b>	<b>1</b>
1.1. BeeSecured Project Background.....	1
1.2. Client Profiles .....	1
1.3. Motivation.....	2
1.4. Objective .....	3
1.5. Contribution & Project Scope.....	3
1.6. Report Organization .....	4
<b>2. BeeSecured Project Architecture Overview .....</b>	<b>5</b>
2.1. System Components .....	5
2.1.1. Hardware Components .....	5
2.1.2. Software Components.....	8
2.1.3. Database .....	9
2.2. System Architecture .....	10
2.3. System Functionality .....	12
2.4. Data Flow.....	12
2.4.1. Handling Alarms.....	13
<b>3. Software Requirements and Constraints.....</b>	<b>15</b>
3.1. Uses Cases.....	15
3.2. Constraints.....	17
3.3. Project Schedule .....	18
<b>4. Design Criteria .....</b>	<b>19</b>
4.1. Application Platform.....	19
4.1.1. Platform Research .....	19
4.1.2. Firmware Research.....	21
4.1.3. Database Research .....	22
4.1.4. Map Technology Research .....	23
4.1.5. Method of Deployment.....	24
4.2. Application Type.....	25
4.2.1. Front-End (Client-Side) .....	25
4.2.2. Back-End (Server-Side) .....	26
4.2.3. Communication Method .....	26
4.2.4. Deployment Environment.....	28

4.3.	Database.....	28
4.4.	Database Connector – JDBC .....	29
4.5.	Tools .....	29
<b>5.</b>	<b>Software Architecture .....</b>	<b>30</b>
5.1.1.	User Case Work Flow Scenarios.....	32
<b>6.</b>	<b>Implementation .....</b>	<b>33</b>
6.1.	Implementation Architecture .....	33
6.2.	Design.....	34
6.2.1.	UML Diagrams.....	34
6.2.2.	Client Package.....	35
6.2.2.1.	Classes.....	35
6.2.2.2.	Interfaces.....	38
6.2.3.	Server package.....	38
6.3.	Database.....	39
6.3.1.	Database Design .....	39
6.3.2.	Notifications .....	41
6.4.	EER Diagram .....	42
6.4.1.	Relationship explained .....	43
6.4.2.	Potential Performance Concerns.....	43
6.5.	Testing .....	44
<b>7.</b>	<b>BeeSecured Web Interface.....</b>	<b>45</b>
<b>8.</b>	<b>Future Work .....</b>	<b>50</b>
8.1.	Feature Implementation.....	50
8.2.	Testing .....	51
8.3.	Map Upgrades.....	51
8.3.1.	Google Maps v3.....	51
8.3.2.	GWT Google Map Utilities.....	51
8.3.3.	Mapstraction .....	51
8.4.	Bug Fixes and Code Refactor.....	52
<b>9.</b>	<b>Conclusion.....</b>	<b>53</b>
	<b>References.....</b>	<b>54</b>
	<b>Appendices.....</b>	<b>55</b>
Appendix A.	UML for Client Application .....	56
Appendix B.	UML for Server Application.....	70
Appendix C.	Database Tables .....	72

## List of Tables

Table 2.1: PEG Sensor Table.....	7
Table 2.2: TAG Sensor Table.....	8
Table 3.1: BeeSecured Web Software Requirements.....	17
Table 3.2: Project Schedule .....	18
Table 4.1: Platform Comparison.....	19
Table 4.2: Datastore Storage Comparison .....	22
Table 4.3: Map API Comparison .....	23
Table 4.4: Method of Deployment Concerns .....	24
Table 4.5: SQL database comparison .....	28
Table 6.1: Notifications List .....	41
Table 6.2: Database relationship symbols .....	43
Table 7.1: Device Status Icons.....	47
Table 7.1: Unimplemented Features .....	50
Table C.1: Site Table.....	72
Table C.2: Device Table.....	72
Table C.3: Peg Config Table .....	73
Table C.4: Tag Config Table .....	74
Table C.5: Gateway Config Table.....	75
Table C.6: Alarm Data Table .....	75
Table C.7: External Sensor Type Table.....	76
Table C.8: Gateway Data Table .....	76
Table C.9: Peg Data Table.....	77
Table C.10: Tag Data Table .....	77
Table C.11: Tag Sensor Table .....	78



Table C.12: Device Table .....	78
Table C.13: User Role Table .....	78
Table C.14: Login Table .....	78
Table C.15: Device Type Table .....	79
Table C.16: Sensor State Table .....	79

## List of Figures

Figure 2.1: Lantronix Xport Pro .....	6
Figure 2.2: PEG Device Deployed with Solar Panel .....	7
Figure 2.3: TAG Device .....	8
Figure 2.4: System Architecture of One Site.....	10
Figure 2.5: System Architecture with Multiple Sites .....	11
Figure 2.6: Sequence Diagram for Raising Alarms .....	13
Figure 2.7: Sequence Diagram for Clearing Alarms .....	14
Figure 3.1: Administrator Use Cases.....	16
Figure 3.2: User Use Cases .....	16
Figure 5.1: Concept Diagram of BeeSecured Web.....	30
Figure 6.1: Implementation Architecture Diagram.....	34
Figure 6.2: Class UML Format .....	35
Figure 6.3: EER of BeeSecured Database .....	42
Figure 7.1: BeeSecured Web GUI .....	45
Figure 7.2: Gateway Settings Dialog .....	48
Figure 7.3: PEG Settings Dialog.....	48
Figure 7.4: TAG Settings Dialog.....	48
Figure 7.5: PEG Sensor Status Dialog .....	49
Figure 7.6: TAG Sensor Status Dialog .....	49
Figure 7.7: Device Configuration Dialog .....	49
Figure A.1: Login Class .....	56
Figure A.2: Config Class .....	57
Figure A.3: Constants Class.....	57
Figure A.4: AlarmData Class .....	58

Figure A.5: BeeSecuredWeb Class (Fields) .....	59
Figure A.6: BeeSecuredWeb Class (Methods) .....	60
Figure A.7: GoogleMaps Class (Fields) .....	61
Figure A.8: GoogleMaps Class (Methods) .....	62
Figure A.9: Device, TagDevice, PegDevice Classes .....	63
Figure A.10: DeviceService, DeviceServiceAsync Class .....	64
Figure A.11: DeviceType, ExtSensorType .....	65
Figure A.12: SiteStorage, GwStorage, PegStorage, TagStorage, Site Classes .....	65
Figure A.13: GwStorage Class .....	66
Figure A.14: PegStorage Class .....	67
Figure A.15: TagStorage Class .....	68
Figure A.16: GwConfig, PegConfig, TagConfig Classes .....	69
Figure B.1: Servlet Classes .....	70
Figure B.2: Servlet Classes Continued .....	71

## List of Acronyms

---

<b>Acronyms</b>	<b>Definition</b>
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASP	Active Server Pages
BST	BeeSecured Technologies
DOS	Denial of Service
DBS	Database Server
EC2	Elastic Cloud 2
EER	Enhanced Entity Relationship
GUI	Graphical User Interface
GWT	Google Web Toolkit
HRD	High Replication Datastore
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IDE	Integrated Development Method
IO	Input/Output
JSNI	JavaScript Native Interface
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
OOD	Object Oriented Design
OS	Operating System
PaaS	Platform as a Service
RDMBS	Relational Database Management System
RPC	Remote Procedure Call
SDK	Software Development Kit
SDLC	Software Development Life Cycle
StAX	Streaming API for XML
WSN	Wireless Sensor Network

---

# **1. Introduction**

BeeSecured Web is a web interface designed for an occupational safety monitoring system: BeeSecured. This section will cover a brief introduction to BeeSecured, some of the motivation, objective, and scope for the BeeSecured Web project as well as the organization for the remainder of the report.

## **1.1. BeeSecured Project Background**

The BeeSecured system is an integrated intrusion detection and health monitoring solution that uses a Zigbee wireless sensor network, and it provides an end-to-end solution to monitor for occupational safety. The motivation behind designing a system that utilizes a Zigbee network is to implement a solution that is operationally efficient and low maintenance, and flexible to deploy to various terrains. The system consists of many integral components from hardware, firmware to software. Hardware devices consist of Zigbee coordinators, routers and end-devices for the WSN as well as external sensors that could optionally be added to these devices. Firmware components consist of customized software that configures individual hardware devices as coordinator, router or end-device. Software components consist of front-end applications that are used to monitor network and sensor statuses as well as servers that translate Zigbee messages.

## **1.2. Client Profiles**

BeeSecured applies wireless sensor network technology to monitor and detect the statuses of individual nodes in the network. This particular system is an innovative solution to monitor a closed proximity of both indoor and outdoor environments; some specific client profiles are as follows:

## **Building Security (Indoor)**

BeeSecured can provide a solution to monitor movement and location inside any region that is within the Zigbee network. Nodes can be installed to monitor a hazardous area within a building to ensure the safety of all personnel through determining their location, body position and heart rate.

## **Security for Mining Sites (Outdoor)**

Mining sites are especially dangerous and BeeSecured can be a good solution to monitor the safety of any personnel in a site. Existing security systems may require miners to carry and swipe an access card to monitor entry and exit, and this may result in unaccounted miners in or out of a site when one forgets to swipe the card. The BeeSecured system removes this concern by replacing the swipe card with a wearable TAG node, and automatically detecting when miners enter or leave the site.

## **1.3. Motivation**

The motivation behind the BeeSecured Web project is to allow users easy access to the front-end application that controls and monitors the wireless sensor network. In the previous iteration of the system, a desktop application was implemented; however, the lack of mobility with the application, and the additional effort required for deployment led to the idea for a web front-end. The initial motivation behind BeeSecured Web was to create a cloud application. As there are emerging cloud development platforms, deploying an application into a cloud environment seemed like a very attractive options to potential clients. However, due to various system complications and security concerns, the decision was made to not continue with this approach, and instead proceed with a local, more controllable deployment method.

My motivation to take on this project is to learn more about web development as well as the Zigbee wireless sensor network and have exposure to all the components that make the system possible from hardware, firmware to software. The BeeSecured project has grew substantially since its initial research phase and it was really exciting to be a part of a project with real application scope and potential.

## **1.4. Objective**

The objective of BeeSecured Web is to deliver a user friendly update to the current desktop application with the use of open sourced maps and database technologies. The web application has to be in line with common web standards, as well as software requirements drafted as the project scope. The objective is to deliver a robust application in a short timeframe utilizing open-source map APIs.

## **1.5. Contribution & Project Scope**

The project scope is defined by a set of software features agreed on throughout the agile software design iterations. Features are prioritized by importance in the implementation timeline to accommodate for potential feature completion estimation errors. Exact feature requirements are defined in section 3.

In order to design an appropriate interface for a WSN, understanding how WSNs work is a must. Additional research was done on communication networks and protocols, and in particular, research on the Zigbee protocol standards, as well as encryption, authentication, routing and other security issues contributed to the understanding of the full capabilities of the BeeSecured WSN. Knowledge of WSN security, such as routing security, and the understanding of various types of passive and active attacks further contributed to deciding on an appropriate approach to working with a WSN. From a software and database perspective, extensive knowledge from capabilities of a particular framework, language, SDLC and best practices for designing and working with RDBMS also were also applied to this project. Moreover, the understandings of software concepts such as OOD and web application and server technologies were critical to the design of the application. Furthermore, research on cloud computing were used to make decisions to balance between the business needs of the application and the technical constraints and costs, such as available platforms, development time, capability and security concerns for deployment. Additional analysis on previous versions of the BeeSecured GUI was also done for designing an interface that provides a similar overall experience which allows a seamless transition to the new interface for the end user.

## **1.6. Report Organization**

The rest of the report will be organized as follows: The overall BeeSecured system architecture will be presented in section 2; use cases, software requirements and features will be listed in section 3; BeeSecured Web design criteria are covered in section 4; software architecture and conceptual designs are covered in section 5; implementation details are presented in section 6; BeeSecured Web interface overview is in section 7; future work is discussed in section 8; and finally section 9 concludes.



## **2. BeeSecured Project Architecture Overview**

There are several components that make up BeeSecured, and this section will cover all the components within the system, as well as the overall system architecture and functionalities.

### **2.1. System Components**

The BeeSecured project consists of several hardware and software components. Each device is configured differently by firmware to configurations of a Gateway, PEG, or TAG devices. An additional edge device is required for the Zigbee network to communicate with Ethernet devices, and in our system, we use the Lantronix Xport Pro portable network server running the Evolution operating system.

#### **2.1.1. *Hardware Components***

Hardware components (excluding the Xport Pro below) operate at ranges from -40 C to +85 C, giving the system great versatility for deployment over various terrains and weather conditions.

#### **Edge Device (Lantronix Xport Pro)**

The Xport Pro network server is an embedded networking device that acts as a compact server. It is easy to setup, provides security options for data transfers, and supports an abundance of network protocols comprising of TCP/IP, and HTTP. Our main use of the Xport Port is to forward serial data from the Zigbee network to Ethernet. Once Zigbee messages have been converted and sent via serial port, the Xport Port then forwards this data via TCP/IP to the BeeSecured Server to process and store into the database.

**Figure 2.1: Lantronix Xport Pro**



*Note.* BST (2011); used with permission. Lantronix Xport Pro is the highlighted device.

## **Gateway Devices**

Gateways are programmed as Zigbee coordinators to form and coordinate data in the Zigbee network. Certain Gateways are also programmed to convert Zigbee messages to standard serial data to be sent over an Ethernet connection. Figure 2.1 shows a Gateway device inside a long range WiFi router.

## **PEG Devices**

PEGs are Zigbee routers whose main purpose is to route data from coordinators to the end devices. They also have the capability of adding external sensors (infrared, motion, etc) which enhances the flexibility and coverage the BeeSecured security system. PEG devices have 16 signalization pins that can be used to connect to additional output devices based on sensor triggers, and PEG devices usually stay stationary after system deployment. Table 2.1 lists the available sensors on a PEG device.

**Figure 2.2: PEG Device Deployed with Solar Panel**



Note. BST (2011); used with permission

**Table 2.1: PEG Sensor Table**

PEG Sensors	Description
External sensor A (optional)	Extendible to external sensors such as infrared, motion, magnetic
External sensor B (optional)	Extendible to external sensors such as infrared, motion, magnetic
Vibration sensor	Sensor to detect physical tempering of the device
Temperature sensor	Sensors to detect abnormal temperature conditions that may hinder device operations

### **TAG devices**

TAGs are the end devices in the Zigbee network, and they are wearable nodes given to the person being monitored. They are the most power efficient devices in the system because they do not need to be operating the entire time. TAG devices also have the capability to monitor heart rate via ECG and body positions via accelerometers and can alarm the system if a wearer suffers sudden physical injury or goes into cardiac arrest.

**Figure 2.3: TAG Device**



Note. BST (2011); used with permission

**Table 2.2: TAG Sensor Table**

<b>TAG Sensors</b>	<b>Description</b>
Accelerometers	Sensors to detect relative positions of the Tag wearer
Heart rate	Monitors wearer heart rate for any abnormal heart conditions

## **2.1.2. Software Components**

### **BeeSecured Server**

Previous versions of the Zigbee server processes Zigbee messages from the network and sends them via a proprietary message format to the front-end client application. In order for the server to work with BeeSecured web, it has been upgraded to convert serial bitstream of Zigbee messages, and process them to determine appropriate modifications or updates on the database. The server provides a HTML interface for the administrator to analyze the network for any issues.

### **Front-end GUI**

The front-end GUI is what the user will interact with the BeeSecured system. There are two applications that act as the front-end UI: BeeSecured Client, and BeeSecured Web. BeeSecured Client is a WPF desktop application for the user to monitor the sensor network. It offers a rich UI using the .NET 4.0 Framework and provides audio and visual feedback on the sensor network. BeeSecured Web is the new

light weight counterpart to the desktop application that will run on any web browser, and it will interact primarily with a remote database that holds all sensor network information.

### **Zigbee simulator**

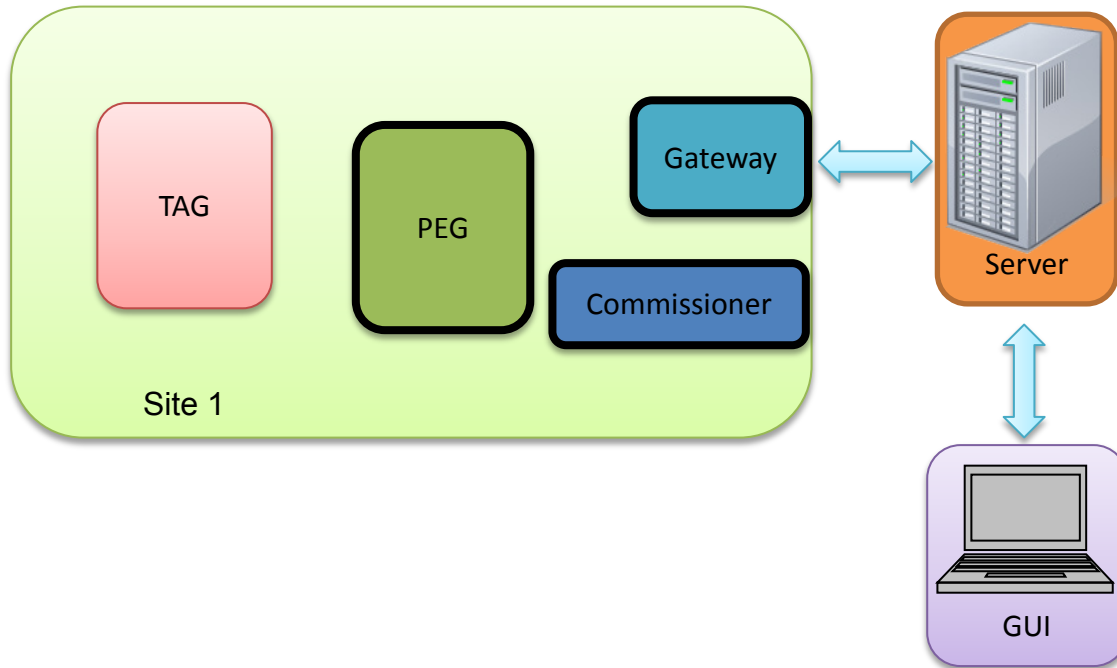
The Zigbee simulator is an application created for generating Zigbee data to simulate the sensor network when no hardware devices are available for testing the software applications. The simulator will periodically generate database data for the front-end GUI to interact with.

### **2.1.3. Database**

In order to maintain a history record of all the data in the sensor network, and login credentials for all users, we use a database to maintain these records. There are several critical features we looked for in selected a database system, which will be discussed in section 4. The database system we use in BeeSecured is SQL.

## 2.2. System Architecture

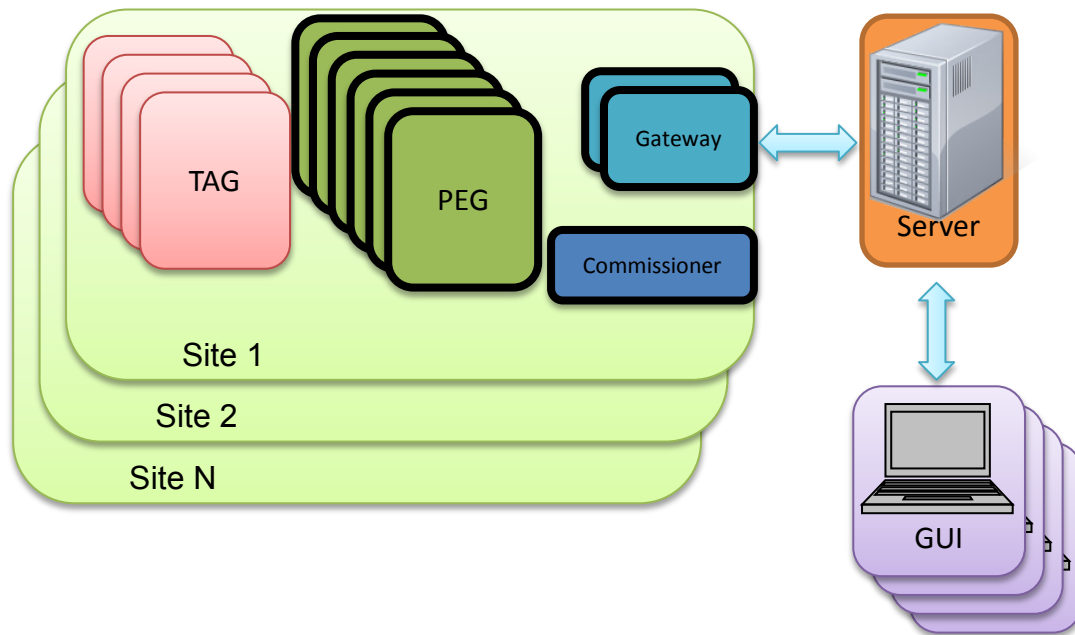
*Figure 2.4: System Architecture of One Site*



*Note.* BST (2011); used with permission

Figure 2.4 shows the system architecture of all the components within a single site within the WSN. Each site consists of a Gateway, PEG, TAG and Commissioner, and the data collected from these components are forwarded to a server and then displayed onto the GUI. The communication channel between the site, server, and GUI are all bidirectional.

**Figure 2.5: System Architecture with Multiple Sites**



*Note.* BST (2011); used with permission

Figure 2.5 shows the architecture of multiple sites within the WSN. The BeeSecured system supports multiple sites, each with its' own set of Gateways, PEGs and TAGs as well as the potential of multiple GUIs. Multiple gateways can be added to the system to increase reliability and multiple GUI access points add flexibility for the end user to monitor the system.

## 2.3. System Functionality

The BeeSecured system provides an end to end solution for monitoring WSN. The current system comprises of the following functionalities:

- Multiple sites in a system
- Very large sites with 500-1000 persons per site (in progress)
- Fixed routers and tags mobile within the site (across multiple subnets)
- TCP/IP connection between sites, server, and GUIs
- Localization (in progress)
- Accounting (in progress)
- Man down detection (physiological monitoring + activity detection)
- Routers with sensors
- Periodic reporting of sensor parameters
- Real-time reporting of alarms
- 3 months battery life for PEG device with no sensors and no solar power
- 5 days battery life for TAG
- Operation -40°C to +80°C and IP67 certification

Note that several functionalities marked as “in progress” are currently not available in the system, and will be included in future implementations of the project.

## 2.4. Data Flow

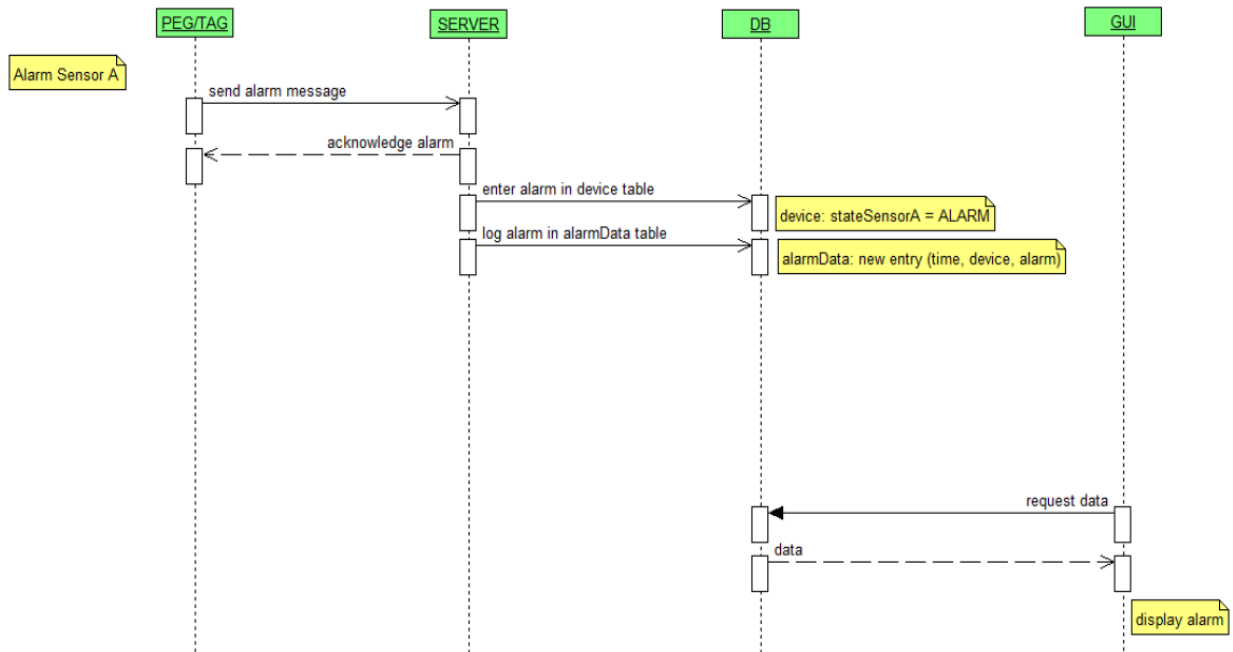
As we can see from Figure 2.4 above, sensor data is collected through PEG and TAG devices, and they are routed through PEGs and Gateways. The data then becomes serialized and forwarded via TCP/IP to the Zigbee server. The server consists of a database, and interprets this data and decides the appropriate actions to act on the database. When the database is updated, the GUI will display these updates to the user. Sensor status data will be sent periodically, and frequency can be set by the administrator. On the event of an alarm, alarm messages will be sent asynchronously through the network, and we will address this below.



### 2.4.1. Handling Alarms

As BeeSecured relies on alarm events to notify end users on intrusion or other critical statuses in the system, the sequence of raising and clearing alarms generated from sensors throughout the system is an important data flow to monitor. The diagram below shows how each component does in the event of an alarm.

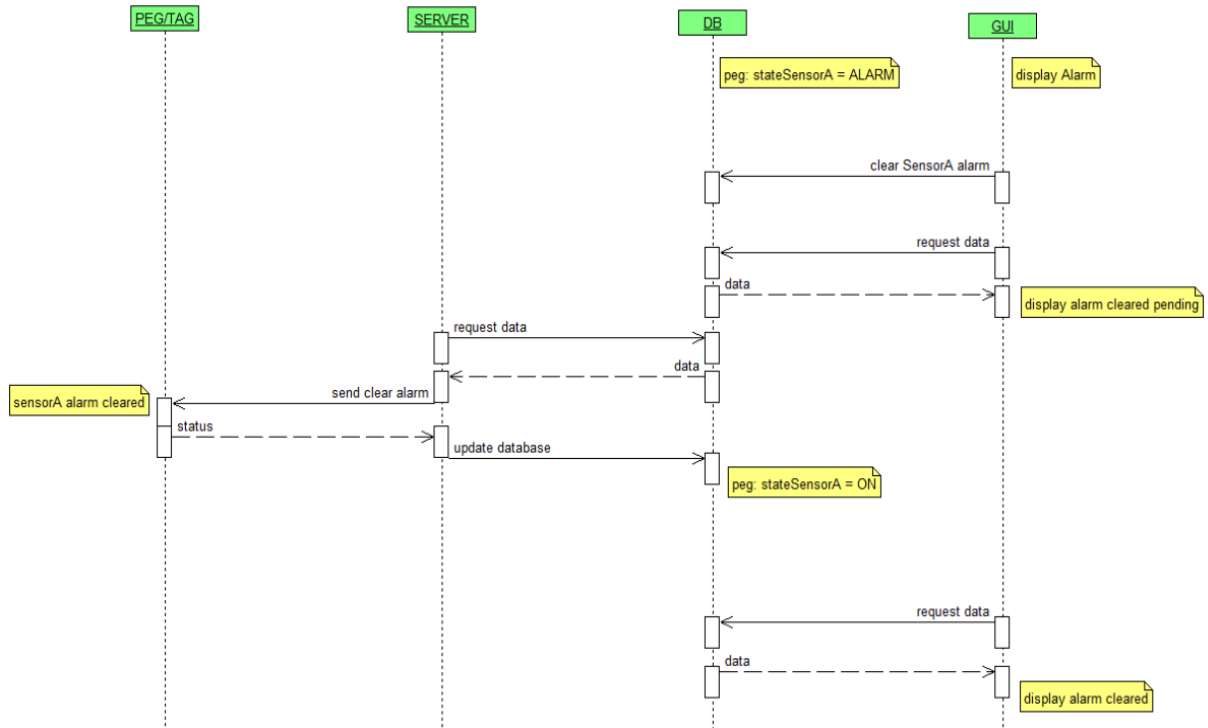
**Figure 2.6: Sequence Diagram for Raising Alarms**



*Note.* BST (2011); used with permission

As we see in figure 2.6, when the system raises an alarm, it sends an alarm message to the server. The server then sends an acknowledgement message back once it receives the alarm. At the same time, the server updates the alarm status in the database and inserts a new entry in the alarmData table as a log. The GUI periodically checks whether the database has an updated alarm status and displays alarm conditions accordingly.

**Figure 2.7: Sequence Diagram for Clearing Alarms**



Note. BST (2011); used with permission

The sequence to clear alarms starts by the user noticing the alarm status on the GUI. By clearing the alarm in the GUI, the GUI sends a clearing alarm pending status to the database. On the event that the database alarm status is updated, the server reads from the database and sends out a clear alarm message to the device(s). Once the device(s) clears the alarm, a status message is sent back to the server, which then writes the updated status to the database. The GUI reads the updated clear status from the database and displays it to the user accordingly.

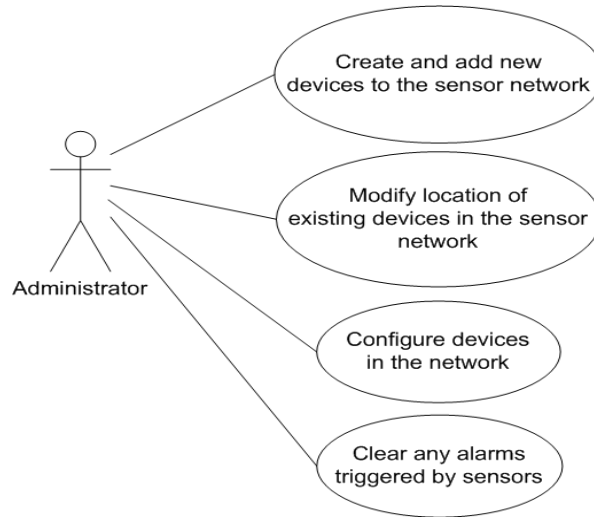
### **3. Software Requirements and Constraints**

BeeSecured Web aims to satisfy two distinct user profiles, commonly used in standard IT security scenarios: the administrator, who has full privilege to modify settings in the application and data in the database; and the standard user, who only has access to monitoring sensor network statuses. In security systems, the standard user typically takes the role of a security guard, who may or may not have extension experience with software applications, therefore, BeeSecured Web must be designed with this consideration in mind. In addition, there are also requirements for integrating the application with a database for storing a unified copy of all sensor network data. The requirements are determined by the uses cases for the individual roles we mentioned, and they are created to cover scenarios that the actors in the systems will encounter. These scenarios are shown in the section 3.1.

#### **3.1. Uses Cases**

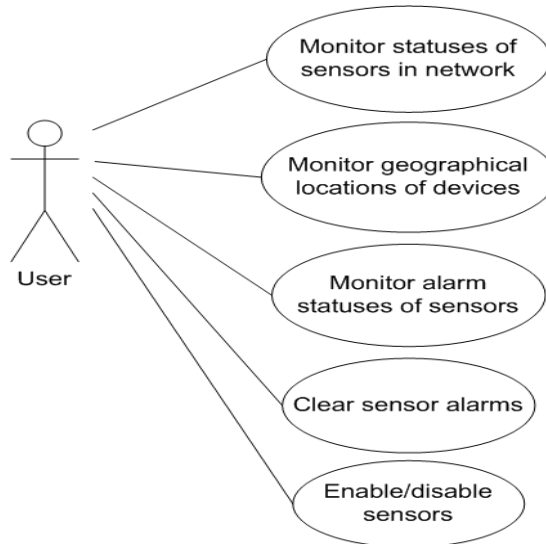
Use cases need to be defined prior to designing software to fit a particular user profile or scenario. Based on the two targeted roles, the fundamental use cases are established and used to map out specific functionalities within BeeSecured Web.

**Figure 3.1: Administrator Use Cases**



*Note.* Administrators have targeted features focusing on the configuration and troubleshooting of the system.

**Figure 3.2: User Use Cases**



*Note.* Standard users have features targeting monitoring the system.

From the uses cases above, the following list of software requirements were drawn for BeeSecured Web:

**Table 3.1: BeeSecured Web Software Requirements**

<b>Feature</b>	<b>Target User Profile</b>
Web application must be able to communicate with a server	Admin/User
Application must have a map that shows site locations via GPS coordinates	Admin/User
Display devices on the maps: separate icons for gateways, pegs, tags	Admin/User
Map must be able to pan and zoom	Admin/User
Login capability with different user profiles	Admin/User
Databases interaction	Admin/User
Configuration of devices	Admin
Enable/disable sensors	Admin/User
Set signalization values for peg devices	Admin
Display alarms (visual and acoustic)	User
Display current sensor data	User
Display sensor data history	User
Display alarm data history	User
Acknowledge/clear alarms	User
Place devices on the map	Admin
Replace/reconfigure devices on the map	Admin
Select a region on the map display present devices	User
External database configuration file for application	Admin

### **3.2. Constraints**

The resources available for this application is limited to one person, and the scope of the project will be limited to the software requirements established. The time constraint is one semester's timeframe of 13 weeks, distributed to research time, learning the required technology, as well as application development. Basic unit testing

is covered throughout the development cycle, but a full test plan will not be implemented or executed. The software life cycle used throughout BeeSecured Web is the agile model, with weekly milestones and reviews with design modifications reduce the risk of a feature not working as originally intended.

### **Agile Model**

- Short development milestones
- Requirements may be updated or changed during development
- End of development depends on feature completion

## **3.3. Project Schedule**

Based on the limited time allocated for the development of the project, BeeSecured Web started from the requirements phase and carried out until the beginning of the testing phase. Full testing phase was not be allocated to the project schedule, nor was maintenance. Minor maintenance may be done post-mortem, but will only go until the end of the year. The following timeline is an approximation of the project timeline throughout development:

***Table 3.2: Project Schedule***

<b>Agenda</b>	<b>Duration (weeks)</b>
Project scope and requirements drafting	1
Initial research and first requirements scoping	1
Technology learning	2
First prototype	1
Weekly milestones	7
Unit testing and additional fixes	1
Total:	13

The agenda is an approximate time spent in each phase in the project. Due to decision changes throughout the development phase, certain tasks were performed in parallel to make up for time left in the remainder of the project schedule.

## 4. Design Criteria

In order to achieve our objective and design a robust, but simple to use web application, there are many factors to consider: the application platform, database, map technology, potential firmware implementations, and the method of deployment. In this section, we will analyze these individual components in the system.

### 4.1. Application Platform

#### 4.1.1. Platform Research

The initial development direction is for BeeSecured Web to be a cloud application; therefore the choice of an appropriate cloud platform was one of our initial concerns. There are several big players that offer cloud platforms, and the following are the platforms that were taken into consideration in the initial project research:

**Table 4.1: Platform Comparison**

	Google	Microsoft	Amazon
<b>Cloud Platform</b>	App Engine	Azure	EC2
<b>Cloud Service Type</b>	PaaS	PaaS	IaaS
<b>Customization Flexibility</b>	Low	Medium	High
<b>Cloud Computing Cost</b>	Free 1GB for incoming/outgoing daily bandwidth limit	\$0.12 per hour	Free Tier 750 hours (1 <sup>st</sup> year). \$0.12 per hour thereafter (small instance)
<b>Database usage Cost</b>	Free (up to 500Mb/month)	\$9.99 per database for 1GB per month	Free Tier 5GB (1 <sup>st</sup> year), \$ thereafter
<b>Development Platform</b>	Open Sourced (Eclipse, NetBeans)	Visual Studios	Open
<b>Development Platform</b>	Free	Per license	Open

<b>Framework</b>	Java/Python/Go SDK	.NET	Open
<b>Application Type</b>	AJAX	ASP.NET (AJAX)	Open
<b>Language</b>	Java/Python/Go	C#	Open

There are a few key factors that were used to decide on a choice for the application: cloud platform cost, development platform cost, and the speed of development. As with most research projects there is a budget constraint to operation costs, so the costs of the cloud usage and development platforms weighed heavily into the choice of development platform. In addition, there is also a need to maintain software portability in the event that there is a need to migrate the application to a different system.

Amazon's EC2 provides the most flexibility in our cloud deployment, as they offer IaaS, where we are given the option to customize and choose any OS (Linux, Windows, OpenSolaris) for our hardware emulation. This allows us to develop on any platform we choose and use any framework we please. For new users, Amazon does offer free quota for the first year, which is beneficial for the development phase of this project. However, since we are developing a web application, there is very little benefit in using a service that provides high hardware customization ability, as is provided by Amazon's EC2. Although it may be useful to customize cloud instances for different customer profiles, the effort to setup the configuration outweighs the benefits. Microsoft and the .NET framework provides a very rich set of libraries for our application, and their integrated Visual Studio 2010 platform has an abundance of features that will allow us to create rich web applications. Unfortunately, the costs of development licenses for Microsoft products do not come cheap, and neither do their services. The portability of code is also fairly low since we will be restricted to relying on the .NET framework. Lastly, for the Google App Engine platform, we have a permanent free quota for database and computing usage, which is a major advantage over the other frameworks. Development for App Engine can be done on any free open-source development platform such as Eclipse or NetBeans, which makes the choice even more appealing. Although the flexibility of the Google platform is the most limited of our choices because we are developing in a sandbox environment, it does cater to all of our needs for deploying a web application.



Based on the mentioned criteria, the Google platform definitely comes out on top with their offer of free quota for computation and database usage, as well as the open-source development platform that they've integrated into their SDK.

#### **4.1.2. *Firmware Research***

The Lantronix Xport Pro with Evolution OS provides a set of APIs for customizations to the onboard server, as well as CGI scripting capabilities for changing their default web interface. The Xport Pro can be used as a solution to push data onto a web server, and have the web application display contents to the user based on that data. However, it would be difficult for the web application to communicate with the Xport Pro since applications run in closed sandbox environments. A potential solution to enable the web application to communicate with Xport Pro is to have the Xport Pro hosted on the web with a public address. Since the Xport Pro has limited hardware capabilities, this solution can leave the system open to DOS attacks directly on the device. One potential option to discourage DOS is to host a HTTPS server on the Xport Pro.

The proposed architecture of BeeSecured Web is to act as an alternative solution to both BeeSecured Client and BeeSecured Server. This architecture requires the Xport Pro network server to not only forward Zigbee messages from serial to TCP/IP, but to also submit it as HTTP requests to BeeSecured Web. The Xport Pro does provide customization features for this mechanism; however it requires writing customized firmware onto the device. On the Zigbee Gateway, there is also no functionality written to send serial data via HTTP format, so another option would be to change the Gateway firmware for this setup to work. Both these solutions, however, requires additional code changes on the existing BeeSecured Client application, so the final decision was against this system design.

After researching on potential firmware modifications, the team decided it was best to make modifications to the BeeSecured Server application and not move forward with combining it with BeeSecured Web. By keeping the BeeSecured Server as a separate application, there is less risk to changing or breaking the overall system.

### 4.1.3. Database Research

Google provides their database service with App Engine, called the datastore. The datastore is based on Google's proprietary database system, BigTable, and it is not distributed outside of Google, but is offered as a service as part of App Engine. BigTable is not a RDMS, because it was designed for distributed mapping across numerous machines. There are two data storage options in the datastore, High Replication, and Master/Slave configurations, and they are summarized below:

**Table 4.2: Datastore Storage Comparison**

	High Replication	Master/Slave
<b>Cost</b>		
Storage	1x	1/3x
Put/Delete CPU	1x	5/8x
Get CPU	1x	1x
Query CPU	1x	1x
<b>Performance</b>		
Put/Delete Latency	1/2x – 1x	1x
Get Latency	1x	1x
Query Latency	1x	1x
<b>Consistency</b>		
Get/Put/Delete	Strong	Strong
Most Queries	Eventual	Strong
<b>Occasional Planned Read-Only Period</b>	No	Yes
<b>Unplanned Downtime</b>	Extremely rare. No data loss.	Rare. Could lose a small % of writes near downtime (recoverable after event).

*Note.* Google App Engine Documentation (2012)

Both storage options can be accessed through the same datastore API provided through App Engine. The default storage method for the datastore service is High Replication. It is more secure than the master/slave storage because it stores data synchronously across multiple datacenters. There is also a benefit in terms of performance for using the HRD over master/slave; since HRD replicates data over

multiple data centers, users are also not prone to scheduled maintenance downtimes. Of course, these benefits come with a price of roughly 3 times more than master/slave.

#### **4.1.4. Map Technology Research**

BeeSecured Web relies on a map system to display the location of each sensor within the network. One possible solution is to create such a system by manipulating custom images and adding our own functionality, but that additional effort would require substantially more time based on the given the list of application requirements. For web applications, there are many map systems that have open APIs with functionalities that provide a rich user experience. The most commonly known system is arguably Google Maps, and there are other systems such as Microsoft’s Bing Maps and Yahoo! Maps we can also choose to integrate into BeeSecured Web. We compared the following map services:

**Table 4.3: Map API Comparison**

<b>Map Technology</b>	<b>Free version?</b>	<b>Free platform conditions</b>	<b>Enterprise License costs</b>
Google Maps	Yes	Map must be publicly available (not allowed for internal applications). Map implementation can be restricted by login, restrictions cannot be fee based. Currently unlimited; 25,000 map loads per day (starting in 2012).	\$10,000 per year
Bing Maps	Yes	Use on public facing, non-password protected web sites. 125,000 sessions or 500,000 transactions per year.	Usage-based, Known user, or per asset based
Yahoo! Maps	Yes	Map must be free of charge (can be internet or intranet applications for personal or business). Unlimited Map loads. 5000 queries per IP per day for Geocoding services.	None

Table 4.3 lists a subset of conditions for free map services by the listed providers. In terms of costs, both Google Maps and Yahoo! Maps currently have no restrictions on map loads, and are both great cost effective solutions. In terms of flexibility, Yahoo! Maps provides the most robust service agreement at the moment as we can use their services for non-public facing applications. We will discuss in the section 4.2 our choice of map services to include in BeeSecured Web.

#### **4.1.5. Method of Deployment**

Cloud technology is fairly new and there are various concerns with using cloud services. For this project, there are several concerns and complexities in particular that we ran into in regards to deciding on a deployment method.

**Table 4.4: Method of Deployment Concerns**

<b>Concerns</b>	<b>Cloud</b>	<b>Traditional Web Hosts</b>
Data Security (excluding physical tempering)	Relies on service provider	Secure
Data Confidentiality	Uncertain	Secure
Data Communication with Sensors	Requires additional firmware development on edge device	No edge device development required
Deployment Environment Setup	None	Yes

One of the main benefits to deploying an application in the cloud is that it does not require much effort. However, although cloud service providers do support data replication for their database, there is still a concern with extracting that data out for backup or using it in another system. In addition, since the BeeSecured system can potentially monitor personnel health status, there is a concern for data confidentiality if it were stored in the cloud.

Asides from data and environment issues, there is a complication with the dataflow of the system. In order for sensor data to be forwarded directly to the web application, additional firmware needs to be written for the Lantronix Xport Pro network server. With the proposed project timeline for feature requirements, the additional learning curve required for firmware development may block feature development for the application.

After considering data control and development requirements, we decided it was better to internally host BeeSecured Web rather than to deploy it onto the cloud. The initial research gone into cloud development did provide a good insight into future potential possibilities, and also allowed us to choose a great development platform, except that we no longer need to utilize its cloud capabilities.

## **4.2. 4.2 Application Type**

### **4.2.1. *Front-End (Client-Side)***

Web development has come a long way since the first version of HTML and there are many technologies and standards to govern a good application. The design requirements for BeeSecured Web are cross-browser compatibility and platform independency, and if we consider a Java application, one option is to develop a Java Applet as our application front-end. However, in order to Java Applets to run on an OS, the user must download the application and have JVM installed on their machine, which may add unfavorable delays for the end-user. Many web applications are now written in AJAX, which allows asynchronous retrieval of data from the server to the client based on UI interactions. AJAX allows dynamic displays in JavaScript without full page loads, giving the user a smooth experience.

With AJAX, we need to write our client-side application in JavaScript. However, Google has provided a framework used by many Google products such as AdWords and Orkut, called GWT, and it allows us to write our application in Java and compile in AJAX for faster development. With GWT, Google has also provided a very valuable collection of Java wrapper libraries for different Google product APIs (Maps, Charts, Calendar, etc) which makes it simpler to integrate Google products with our application.

Google App Engine supports three languages for development: Java, Python, and Go. Keeping in mind the constraint of open-source development platforms, we selected to use Java because it is the supported language for the GWT framework. One important feature we require for our application is to have a map to display the location of our sensor nodes. There are many maps applications with open APIs available, as shown in Table 4.3, such as Google Maps, Bing Maps, and Yahoo! Maps, and we need

to select one that has JavaScript libraries in the language of our choice. Although in terms of robustness and current future usage speculations, Yahoo! Maps offers the most cost effective service, currently, there is no GWT library available for it. On the other hand, GWT has Java libraries available for many Google products, including Google Maps; therefore, the decision was made for BeeSecured Web to be developed in Java using the Google Maps service.

#### **4.2.2. *Back-End (Server-Side)***

BeeSecured Web requires a server side component because it needs to accomplish certain tasks that cannot be implemented on the client side application, such as directly accessing a database through JDBC and reading configuration files from the server's file system. As with the client-side application, our server application can be done in various languages. The language choice for the server will depend on the type of web container we eventually decide to use in our deployment server.

Since we are using GWT, the obvious choice is to develop the back-end with Java. There are numerous web services that support Java servlets so for the ease of development; BeeSecured Web uses Java Servlets as the back-end.

#### **4.2.3. *Communication Method***

##### **GWT RPC vs. HTTP Requests**

Communication between the GWT client-side and server side application can be done either through GWT RPC or HTTP requests using GWT supplied RequestBuilder class. GWT RPC is a great way for Java front-end applications to talk to the server because GWT takes care of all the object serialization and deserialization for you, as it is an efficient method to serialized objects across networks using deferred bindings. Alternatively, we can communicate with the server by submitting HTTP requests through RequestBuilder if we choose not to use a Java backend. There are several classes in GWT that allows us to write custom HTTP request and we can then process a JSON or XML formatted response. GWT does not limit to these two methods of communication,

as you can also use JSNI methods or third party libraries as other forms of RPC mechanisms.

Although HTTP requests potentially offer more flexibility to our choice in back-end applications, RPC is much simpler to setup as it does not require parsing of request URL or JSON, XML responses. With a Java backend, RPC also offers greater performance; therefore BeeSecured Web uses RPC as the main method of communication between client and server applications.

### **Serializable Types**

In designing classes to work with GWT RPC, we need to be aware of the supported serializable types to ensure data can be successfully passed between the client and server. The list below specifies a subset of the conditions that are of concern to BeeSecured Web.

A type is serializable if one of the following is true:

- The type is primitive, such as char, byte, short, int, long, boolean, float, or double.
- The type is an instance of the String, Date, or a primitive wrapper such as Character, Byte, Short, Integer, Long, Boolean, Float, or Double.
- The type is an enumeration. Enumeration constants are serialized as a name only; none of the field values are serialized.
- The type is an array of serializable types (including other serializable arrays).
- The type is a serializable user-defined class.
- The type has at least one serializable subclass.

*Note.* Google GWT (2012)

### **Serializable User-Defined Classes**

All of the following conditions need to be met for a user-defined class to be serializable:

- Either directly implements `IsSerializable` or `Serializable` interface or derives from a superclass that does.

- All non-final, non-transient instance fields are themselves serializable
- As of GWT 1.5, it must have a default (zero argument) constructor (with any access modifier) or no constructor at all.

*Note.* Google GWT (2012)

#### **4.2.4. Deployment Environment**

With the GWT SDK, a local Jetty development server has been integrated for the use of debugging during development. The choice of our web container is not extremely critical, given that it's stable and reliable, as long as it supports Java.

Apache Tomcat has been around for a very long time, and has proven to be extremely reliable; therefore, we decided to use Tomcat as our servlet container for the deployment of BeeSecured Web.

### **4.3. Database**

Another important component to select is our database. As we decided to locally host our application, we have a range of databases to consider. Our desktop application, BeeSecured Client, utilizes the Microsoft SQL database for authenticating user login information. It will eventually be migrated to an open-source database to keep the system running on a unified database, so in order to keep our selection simple, we decided to continue using SQL as our database system.

In selecting a SQL product, we selected from the following:

**Table 4.5: SQL database comparison**

<b>Criteria</b>	<b>Microsoft SQL</b>	<b>MySQL</b>	<b>PostgreSQL</b>
Commercial Cost	Yes	Free for open-source version	Free
Notifications	No	No	Yes



One important criterion for our database selection is the database must be able to asynchronously notify external applications on particular modifications to a table. This is crucial because instead of a one way communication, where the client application initiates all requests to the server, we need a method for the application to query the database only when there is updated data instead of periodically fetching redundant data. We first decided to use MySQL, however, we then realized it does not support the asynchronous notification events we needed so we migrated to PostgreSQL. Of course, there are many other differences between these database systems, but for the purpose of our application, the complex features that may cater to large complex queries or data replication are not too much of a concern at this stage.

#### **4.4. Database Connector – JDBC**

In order to connect to and interact with our database, we need a database driver for Java. JDBC is an open-source API that abstracts the implementation of Java programs to various different database systems. This allows us to write our database connection code once, and gives us the flexibility to change between different databases as we please. As such, most database systems provide JDBC drivers for their systems, and the PostgreSQL JDBC driver we use in BeeSecured Web is JDBC4 Postgresql Driver, Version 9.1-901.

#### **4.5. Tools**

One of our design criteria is to use free and open-source development platforms. We selected Eclipse as the choice of development tool because GWT offers plugins for the Eclipse IDE. GWT's Jetty local development server is also integrated to provide fast testing during development.

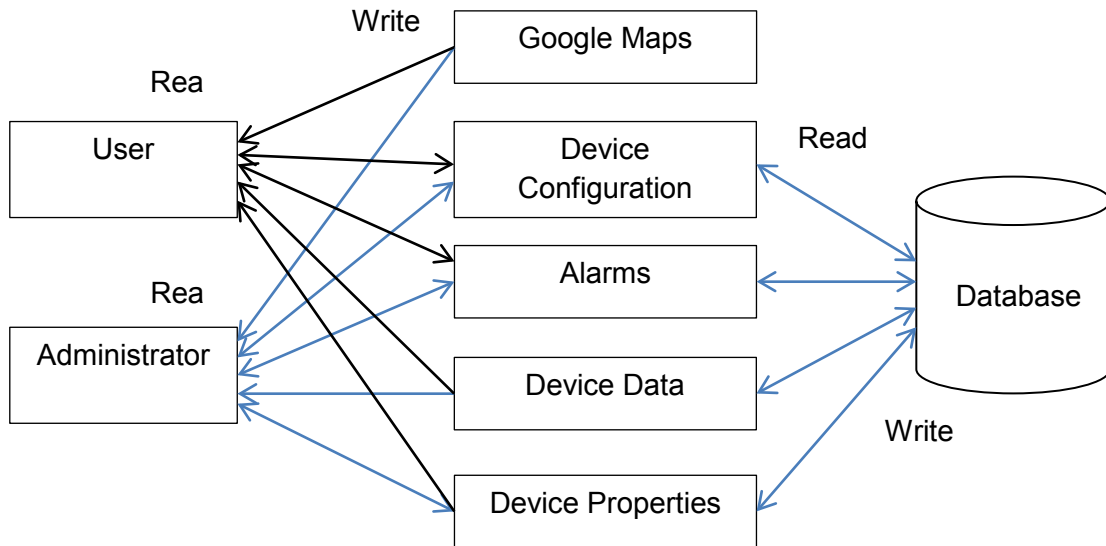
For database tools, PostgreSQL has both a command shell and a graphical user interface, pgAdmin for managing connections the local or remote databases.

## 5. Software Architecture

As we mentioned, there are two user profiles we need to cater to in BeeSecured Web, each with unique access rights to the application. Both profiles have access to the same login page, where no sensitive data is presented. The main interaction with the application is through the Google Maps API, and most of the functionality within the application is tied with features from Maps. All data will be stored in the database, and the application will fetch and update data to display new information on the map either on user request, or on database notification updates.

The conceptual software architecture of the application below shows the interaction with individual functional blocks within the application.

**Figure 5.1: Concept Diagram of BeeSecured Web**



Google Maps block represents all API calls made to Maps. Device configuration block consists of settings that change the behavior of a device, such as status report intervals, alarm delay, and external sensor types. The alarms block consists of all alarm statuses, timestamps and alarm messages. Device data block consists of any periodic data updates from the devices, such as temperature, battery voltage, and accelerometer readings. Finally, device properties represent fields such as geographic positioning of devices, the parent site a device belongs to, and the name of the device.

These functional blocks by no means represent a single class in the application and can encapsulate multiple classes that work together to achieve the block function. Detail implementations of each class will be explained in section 6, and please refer to appendix C for detailed fields in the database table.

Users have mostly read privileges for monitoring the WSN, with the addition that they can enable or disable sensors on any particular PEG or TAG device. Administrators can also monitor the WSN, but have the ability to modify device configurations, as well as other properties available for updates such as device names and their parent sites.

### **5.1.1. User Case Work Flow Scenarios**

A few common work flows are described below to show how a user would typically interact with BeeSecured Web:

#### **Administrator wants to update device configuration**

1. Administrator finds device on the map.
2. Administrator clicks on device configuration menu.
3. Device configurations read from the database and displays in dialog.
4. Administrator updates configurations and commits the changes on UI.
5. Device configuration writes to the database.

#### **User wants to clear an alarm**

1. An alarm is triggered in the WSN database sends out a notification.
2. Alarm updates device status on the map.
3. User finds the devices and clicks the clear alarm option.
4. Clear alarm writes updated status to the database.

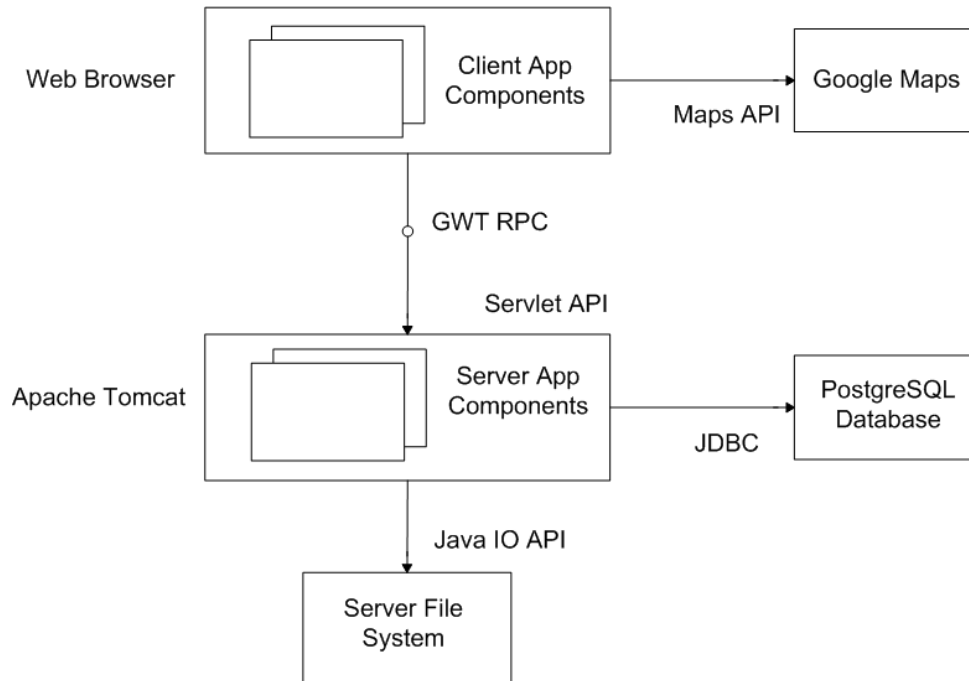
## **6. Implementation**

This section covers the implementation with regards to BeeSecured Web. Detailed descriptions about the implementation architecture, organization of each package in the application and their classes, as well as the design criteria for the database are covered.

### **6.1. Implementation Architecture**

BeeSecured Web is organized into two packages: `com.ciber.beesecuredweb.client`, and `com.ciber.beesecuredweb.server`. The client package consists of Java code that compiles to JavaScript to run in a browser, and the server package compiles to Java `.class` files to run on the Apache Tomcat web container. The client package interacts with Google Map's API, as well as the servlet's GWT RPC interface. The server package interacts with the PostgreSQL database using JDBC, as well as the file IO interface for reading configuration files from the server's file system. Figure 6.1 shows the implementation architecture of BeeSecured Web.

**Figure 6.1: Implementation Architecture Diagram**



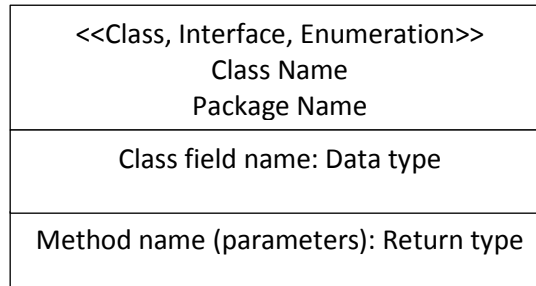
## 6.2. Design

### 6.2.1. UML Diagrams

This section describes in detail each class in BeeSecured Web and their relationships within the program. The UML diagrams will include any fields, methods and relations to other classes. Since there are many classes in the program, they will be discussed separately. Classes will be explained in a standard user work flow, starting with the client package, then moving on to the sever package.

UML diagrams will have the following format:

**Figure 6.2: Class UML Format**



Due to the size of the diagrams, please refer to appendix A for full class UML regarding all the classes in the com.ciber.beesecuredweb.client and com.ciber.beesecuredweb.server packages.

## **6.2.2. Client Package**

Description of each class and their usage within the program will be described below. The general organization and structure of classes is based off of the database design, as we encapsulate all the table columns as variable fields in the class objects.

### **6.2.2.1. Classes**

#### **Login**

Login class is used for authentication user credentials on accessing sensor network data. The class encapsulates all the data fields in the login database table as well as providing an enumeration of login statuses for the application.

#### **Site**

Site class is defined to store properties such as the name and location of a particular site in the system.

## **Config**

The Config class is to store properties for the configuration files that hold the database connection parameters which BeeSecured Web reads on application load.

## **Constants**

All global constants used throughout the application are defined in this class.

## **AlarmData**

The AlarmData class encapsulates all properties of alarm events to pass between the client and server application.

## **Device**

Device class holds all device properties to a particular device. It is the base class for all Gateway, PEG and TAG devices. Since Gateways do not have sensors, they are represented as the Device object in the application.

## **PegDevice, TagDevice**

These classes are child classes of the Device class, and hold additional fields and methods such as sensor fields that are particular to a PEG or TAG device.

## **DeviceType**

Device type class is used to hold the names and IDs for the types of devices in the system. Current devices in the system are fixed to Gateways, PEGs and TAGs.

## **ExtSensorType**

This class holds all external sensor types properties for optional sensors that could be attached to PEG devices.



## **GoogleMaps**

The GoogleMaps class handles all API calls to the Maps API, along with any methods that keeps track of the most updated data in the application.

## **BeeSecuredWeb (Entry Point)**

BeeSecuredWeb is the entry point to our application. It holds fields and methods that manipulate and display the user interface, and interacts closely with the GoogleMaps class. This class is also responsible for handling user password encryption for the application.

## **GwConfig, PegConfig, TagConfig**

These classes hold all configurations pertinent to each device type. Configurations are parameters customizable by the user that govern how the device operates, such as alarm delays, or status reporting intervals.

## **GwData, PegData, TagData**

These classes encapsulate all information regarding the status of each device. Data readings such as temperature levels, battery levels, and accelerometer values are fields in these classes.

## **GwStorage, PegStorage, TagStorage**

The storage classes are wrapper classes that hold the device properties, data and configuration of a particular device. These classes are created to encapsulate all information regarding a particular device as a simple form to storing the most update to date information in the application.

## **SiteStorage**

The SiteStorage class holds lists of all the device storages (GwStorage, PegStorage, TagStorage) for a particular site. The class allows a unified object to manipulate for individual sites in the application.

#### **6.2.2.2. Interfaces**

##### **DeviceService**

The DeviceService interface defines all methods that are used on the server application. This interface is accessed to use GWT RPC to pass java objects between client and server applications.

##### **DeviceServiceAsync**

This class is defined by GWT RPC and is an asynchronous interface equivalent to the DeviceService interface. It contains all the same methods in the DeviceService interface except all methods have a return type of void.

#### **6.2.3. Server package**

##### **dbConnection**

dbConnection class is used for create new JDBC connections based on the parameters read in the local configuration file.

##### **DeviceServiceImpl**

DeviceServiceImpl implements the DeviceService interface in the client package, and provides the implementations of methods that submit different SQL commands to PostgreSQL once it receives RPC calls from the client.

##### **Log**

The log class uses the Java file IO API to write to a separate log file when there are connections or SQL exceptions. The purpose of this class is to provide an easy way for the administrator to debug the application if errors occur.

##### **notificationListener**

BeeSecured Web periodically checks for notifications coming for PostgreSQL on table updates, and the notificationListener class spawns a separate thread on the server

to maintain a connection to the database; this thread will spawn on application load, and will terminate on application exit.

## **StAXParser**

The StAXParser class is used to parse the configuration XML that stores the database connection strings using the StAX API. The reason for using StAX is because it allows both pulling and pushing of XML data.

## **6.3. Database**

The database used with BeeSecured Web is PostgreSQL, an open source RDBMS under the PostgreSQL license. The license entitles you to distribute, modify and make any enhances to PostgreSQL as you like. PostgreSQL is not as powerful as other DBMS and is not capable of running enterprises, as it is a platform for in-house development that may require RDBMS capabilities.

BeeSecured's database server can be used by several customer networks, and each customer uses a separate database with custom access privileges. This delivery method is beneficial for the customer such that they do not need to maintain a database, and can monitor their sensor networks through BeeSecured Web via any web browser using the credentials we've provided them. Alternatively, a customer can use their own private DBS and still deploy the same web application, since BeeSecured uses external XML configuration files to access databases and Google Map APIs. If the customer does choose to employ their own server, they still need to abide by Google's conditions for using their map services and expose the application to the public. This would not be an issue in terms of sensor network data security however, since BeeSecured Web relies on login credentials to access any proprietary information.

### **6.3.1. Database Design**

The SQL database is designed based on the three orders of normal forms of RDBMS design. The database model minimizes duplicate information and is designed

to minimize potential anomalies from user input. Due to the length of the database tables, please refer to appendix B for the full database tables and their descriptions.

### **Key choice criteria**

- Minimality: choose fewest columns
- Stability: column that seldom changes
- Simplicity: simple and familiar to users

### **Purpose of database design**

- Efficient data entry, update, deletions
- Efficient retrieval, reporting (query calculations)
- Self-documenting
- Changes in schema is easy to make
- Prevent anomalies

### **Normal Form design in tables** (the higher order the better, more efficient the design)

#### First Normal Form (1NF)

- Only one value per row-column (atomic)
- No repeating groups in columns

#### Second Normal Form (2NF)

- Is in the First Normal form
- Every non-key column is fully dependent on the (entire) primary key. Entire here means if your primary key is composite, all other columns depend on the composite key, and not just part of it.

#### Third Normal Form (3NF)

- Is in Second Normal form
- All non-key columns are mutually independent

### 6.3.2. Notifications

In order to prevent needing the client application to consistently query the database for new data when there are no updates, we need an asynchronous method to notify the client application when there is an update to the database. PostgreSQL supports asynchronous notifications on table events that allow external applications to register to handle these events when they fire. The table below summarizes all notifications supported in BeeSecured Web.

**Table 6.1: Notifications List**

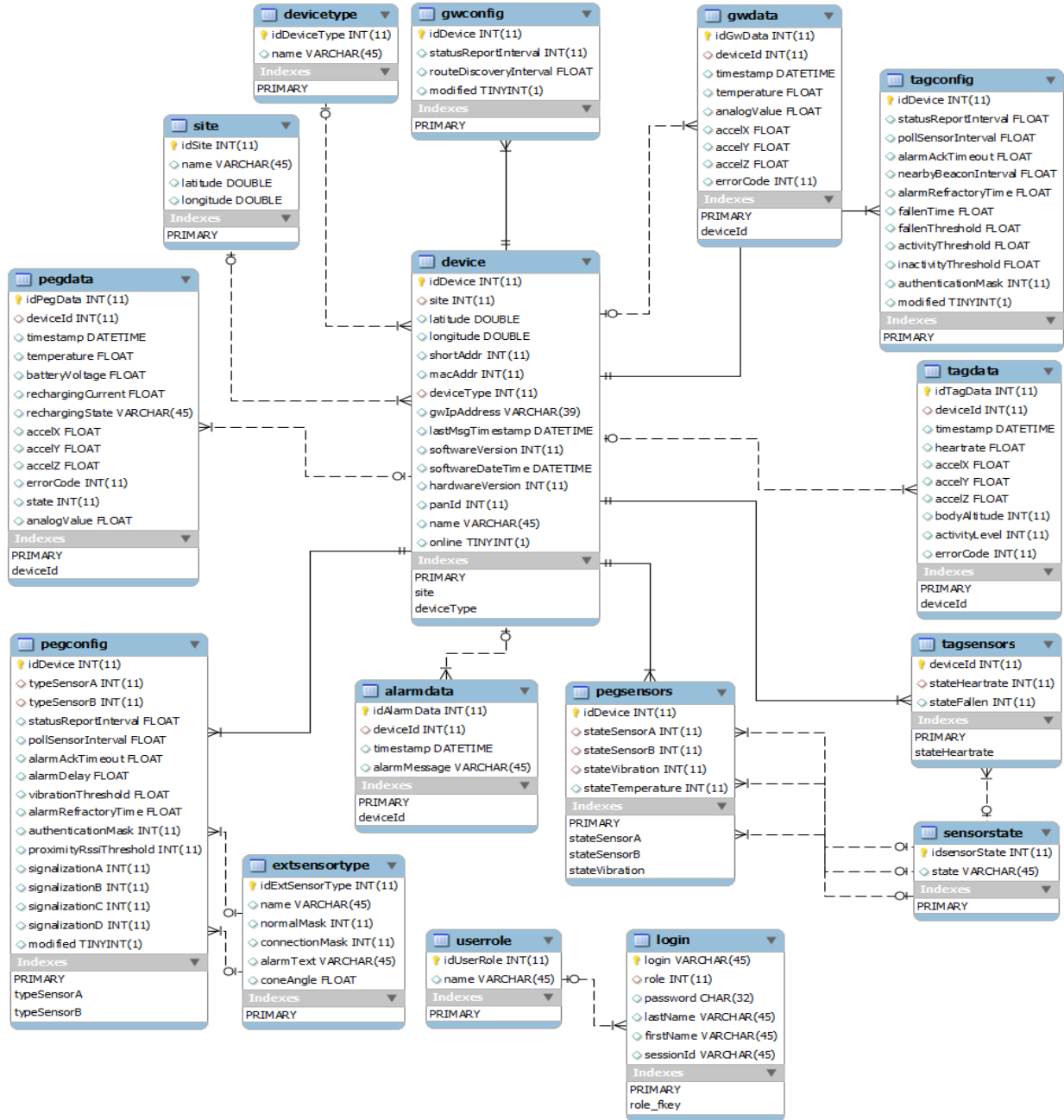
Table	Event	Notification
gw_data	INSERT	gwDataTableInsert
pegData	INSERT	pegDataTableInsert
tagData	INSERT	tagDataTableInsert
pegSensor	UPDATE	pegSensorTableUpdate
tagSensor	UPDATE	tagSensorTableUpdate

Note. BST (2011)

JDBC currently does not support asynchronously notification updates to external applications, as the implementation has not been completed. In order to utilize the notifications feature, BeeSecured Web spawns an alternative thread on the server to periodically check for notification updates coming from the database. The client application periodically make RPC calls to the server to obtain the most updated notifications list and updates the user interface accordingly.

## 6.4. EER Diagram

Figure 6.3: EER of BeeSecured Database



Note. BST (2011); used with permission

Figure 6.1 shows the EER for the BeeSecured database, and shows the relationship between each table.

### 6.4.1. Relationship explained

**Table 6.2: Database relationship symbols**

Symbols	Description
>-----o	N : 1 non-identifying relationship
>-----	N : 1 identifying relationship

An identifying relationship means the child table can be uniquely identified (can exist) without the parent table; a non-identifying relationship means the child table cannot exist without the parent table. A many-to-one (N : 1) relationship means there can be multiple rows in the 'N' table for each row in the '1' table, or in other words a foreign key is added in the 'N' table, and it is constrained to the primary key of the '1' table. As an example of the non-identifying relationship, a device can exist without defining a site. An example of the identifying relationship: a peg (device) configuration cannot exist without defining a device.

In a hierarchical sense, the topmost layer is the site table, and in each site, there can be many devices. A device can either be a Gateway, Peg or Tag, and each of them has configurations, data, and sensors.

### 6.4.2. Potential Performance Concerns

Since this database is not too complex, there is little concern over queries that require joining multiple tables. In BeeSecured Web, the most join required for fetching data is limited to 4 per query. The only performance concerns may be with the gwData, pegData, tagData and alarmData tables. As these tables store a history of all incoming data from the sensor network, they will grow over time and BeeSecured Web may suffer performance issues when it needs to query for the most updated data. To guard against potential performance degradation, the data from these tables may need to be backed up and cleaned up on a periodic basis.

## **6.5. Testing**

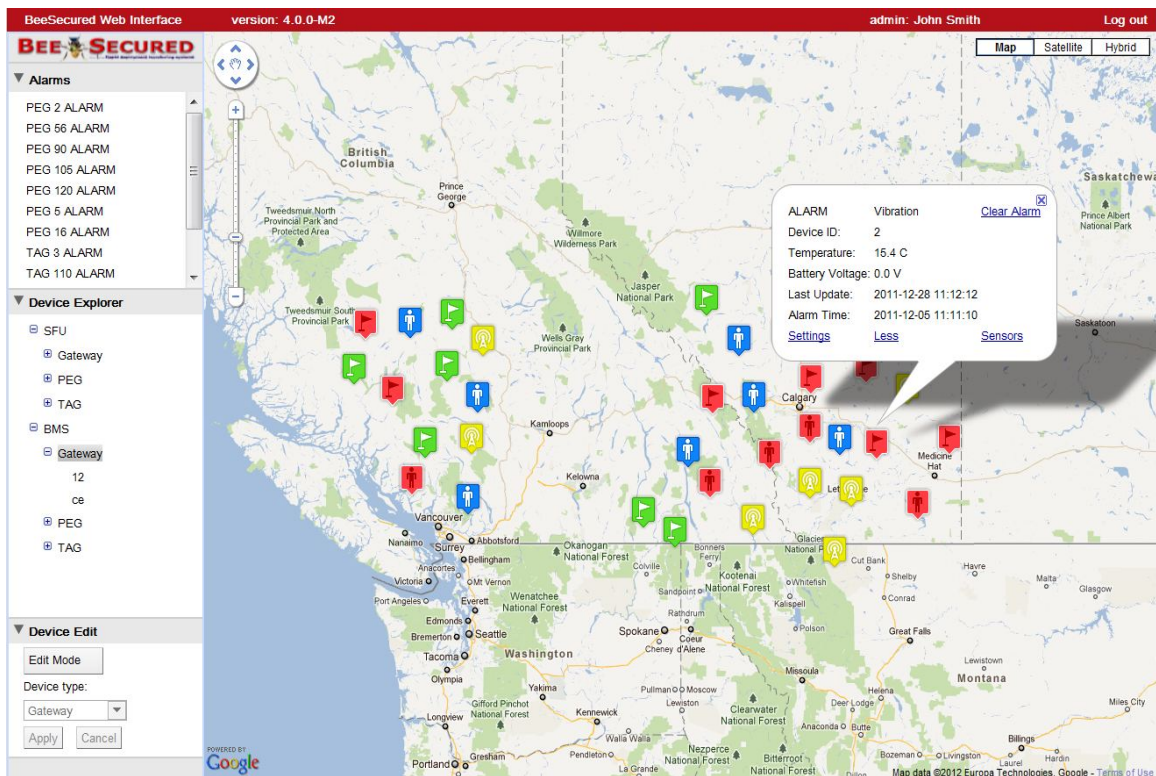
Due to time constraints, no formal testing schedule has been planned throughout the project; however, each individual feature has been tested via Ad-hoc functionally tests. Regression testing on individual features has been done throughout the project to ensure all newly added functionality work with previously implemented features. Because there are not enough physical devices manufactured at this stage to construct a proper WSN, majority of testing has been done using the Zigbee simulator, injecting artificial data into the database to simulate a functional sensor network. The purpose of testing with the simulator is to get a sense of the user experience through using BeeSecured Web.



## 7. BeeSecured Web Interface

This section will discuss the layout of the BeeSecured Web interface. The interface design was aimed to provide most of the functionalities of the application within Google Maps, with an additional left side panel that gives the user additional information and options.

**Figure 7.1: BeeSecured Web GUI**



*Note.* This interface displays the full functional view available to administrators. Standard users will have a more limited view to the application.

The left panel of the interface holds three separate disclosure panels that allow the user to hide or expand the information based on their needs. The three panels are as follows: Alarm, Device Explorer, and Device Edit.

### **Alarm Panel**

The Alarm Panel shows any alarm that has been triggered by the database. As notifications are generated from the database, the Alarm Panel displays any outstanding (uncleared) alarms that are currently in the system. Each item in the panel is also clickable, and directs the user to the corresponding icons displayed on Google Maps.

### **Device Explorer Panel**

The Device Explorer Panel holds all the devices in the network and categorizes them by their site, device type and device IDs. This panel offers the user a convenient way to find a specific device if they know the device ID. In most cases, users will likely monitor devices directly on the Map, so this panel acts as a supplemental view for users to quickly find a particular device.










### **Device Edit Panel**

This panel is only available to the administrator profile, as it contains most of the configuration options to the application. In Edit Mode, administrators have the option to reposition devices geographically on the map, which will update the sensor network to their updated latitude and longitude. Devices can also be renamed, as well as relocated to another site in another network.

### **Top Bar**

The Top Bar is where the user go to log into the application, and it also displays the user name and the current version number of BeeSecured Web. The position of the items in the Top Bar will be moved systematically as the user resizes the browser window.

**Table 7.1: Device Status Icons**

Device Type	Online	Alarm	Offline
Gateway			
PEG			
TAG			

The different device types are represented in the application with their distinct color code as well as icons to allow the user to easily distinguish between them. Devices in the network are represented as in either online, offline or alarm states, and these icons are updated in real-time to reflect the most updated device information in the network. Icons are clickable, and will display a Google Maps InfoWindow that holds all the information to a particular device.

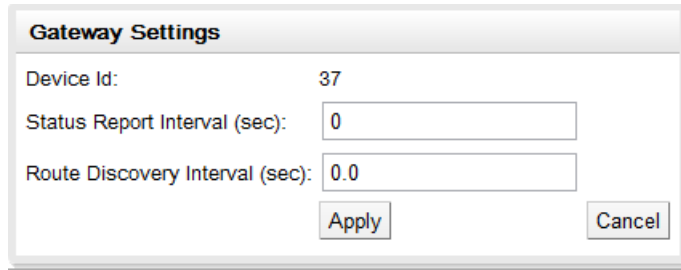
### Google Maps InfoWindow

The InfoWindow in Google Maps is the popup window that display additional icon information when clicked. In BeeSecured Web, the InfoWindow holds the most updated device information from the database, and it also holds controls to configure device settings, clear alarms, and pin or unpin icon locations depending on the application mode. In most cases, users will use the InfoWindow to monitor device statuses and handle any device alarms.

### Configuration Settings

The figures below show all the configuration options available in BeeSecured Web. All the configuration settings can only be accessed through the administrator profile, except for the PEG and TAG sensor status controls that allow a user to enable or disable a sensor conveniently.

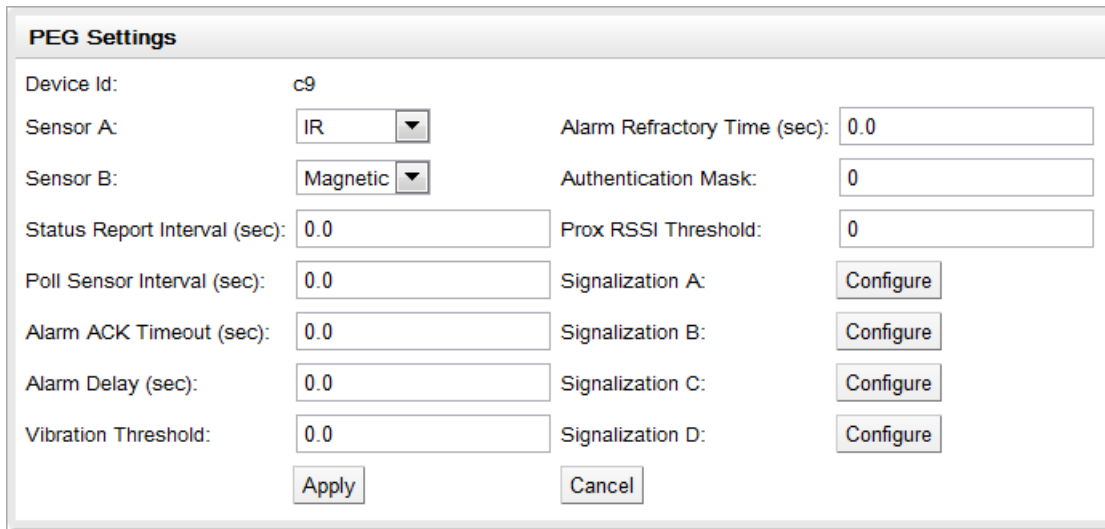
**Figure 7.2: Gateway Settings Dialog**



The Gateway Settings dialog box contains the following fields and controls:

Device Id:	37
Status Report Interval (sec):	<input type="text" value="0"/>
Route Discovery Interval (sec):	<input type="text" value="0.0"/>
<input type="button" value="Apply"/> <input type="button" value="Cancel"/>	

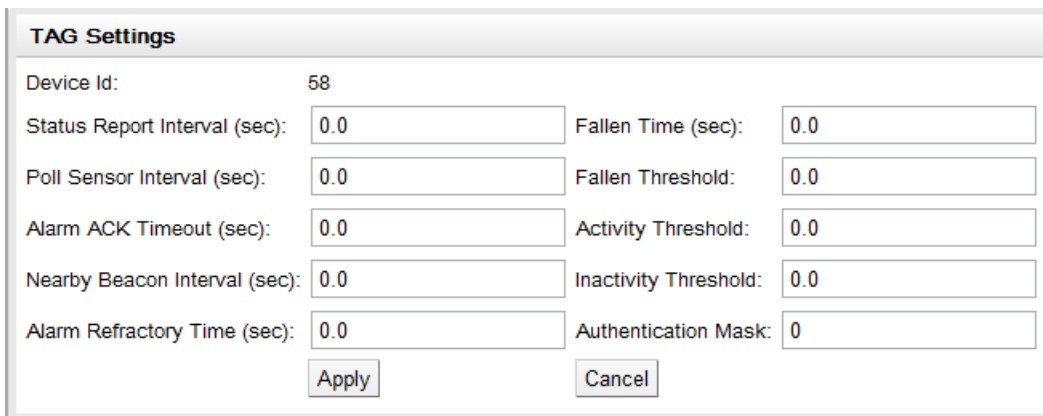
**Figure 7.3: PEG Settings Dialog**



The PEG Settings dialog box contains the following fields and controls:

Device Id:	c9		
Sensor A:	<input type="text" value="IR"/> <input type="button" value="v"/>	Alarm Refractory Time (sec):	<input type="text" value="0.0"/>
Sensor B:	<input type="text" value="Magnetic"/> <input type="button" value="v"/>	Authentication Mask:	<input type="text" value="0"/>
Status Report Interval (sec):	<input type="text" value="0.0"/>	Prox RSSI Threshold:	<input type="text" value="0"/>
Poll Sensor Interval (sec):	<input type="text" value="0.0"/>	Signalization A:	<input type="button" value="Configure"/>
Alarm ACK Timeout (sec):	<input type="text" value="0.0"/>	Signalization B:	<input type="button" value="Configure"/>
Alarm Delay (sec):	<input type="text" value="0.0"/>	Signalization C:	<input type="button" value="Configure"/>
Vibration Threshold:	<input type="text" value="0.0"/>	Signalization D:	<input type="button" value="Configure"/>
<input type="button" value="Apply"/> <input type="button" value="Cancel"/>			

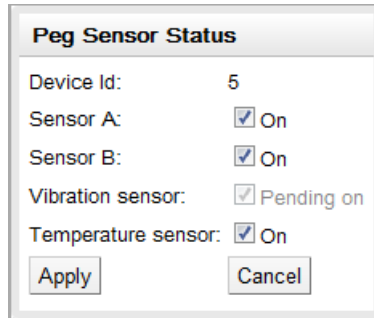
**Figure 7.4: TAG Settings Dialog**



The TAG Settings dialog box contains the following fields and controls:

Device Id:	58		
Status Report Interval (sec):	<input type="text" value="0.0"/>	Fallen Time (sec):	<input type="text" value="0.0"/>
Poll Sensor Interval (sec):	<input type="text" value="0.0"/>	Fallen Threshold:	<input type="text" value="0.0"/>
Alarm ACK Timeout (sec):	<input type="text" value="0.0"/>	Activity Threshold:	<input type="text" value="0.0"/>
Nearby Beacon Interval (sec):	<input type="text" value="0.0"/>	Inactivity Threshold:	<input type="text" value="0.0"/>
Alarm Refractory Time (sec):	<input type="text" value="0.0"/>	Authentication Mask:	<input type="text" value="0"/>
<input type="button" value="Apply"/> <input type="button" value="Cancel"/>			

**Figure 7.5: PEG Sensor Status Dialog**



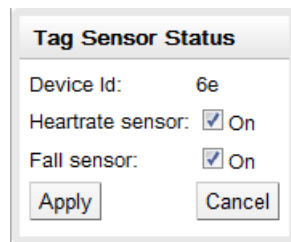
The screenshot shows a dialog box titled "Peg Sensor Status". It contains the following fields and controls:

Device Id:	5
Sensor A:	<input checked="" type="checkbox"/> On
Sensor B:	<input checked="" type="checkbox"/> On
Vibration sensor:	<input checked="" type="checkbox"/> Pending on
Temperature sensor:	<input checked="" type="checkbox"/> On

At the bottom of the dialog are two buttons: "Apply" and "Cancel".

*Note.* Sensor statuses can only be set to the pending (On/Off) states from BeeSecured Web. The Zigbee server will send messages to update the devices in the network, and only when the physical device status has been changed will the GUI display the confirmed updated states.

**Figure 7.6: TAG Sensor Status Dialog**

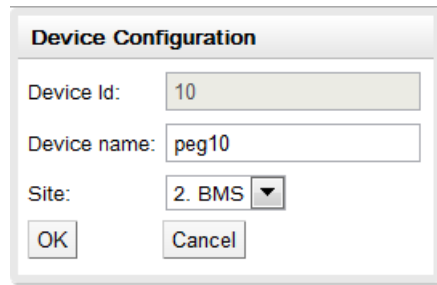


The screenshot shows a dialog box titled "Tag Sensor Status". It contains the following fields and controls:

Device Id:	6e
Heartrate sensor:	<input checked="" type="checkbox"/> On
Fall sensor:	<input checked="" type="checkbox"/> On

At the bottom of the dialog are two buttons: "Apply" and "Cancel".

**Figure 7.7: Device Configuration Dialog**



The screenshot shows a dialog box titled "Device Configuration". It contains the following fields and controls:

Device Id:	10
Device name:	peg10
Site:	2. BMS

At the bottom of the dialog are two buttons: "OK" and "Cancel".

*Note.* The Device ID cannot be changed once it is configured in the application, since the ID is specific to the hardware. If the device is no longer needed in the system, it can be removed in Edit Mode.

## 8. Future Work

Given the short project schedule for BeeSecured Web, there are many areas in the application that could use further work. This section discusses areas of improvements as well as potential new technologies to integrate into the application.

### 8.1. Feature Implementation

There are a few lower priority features that did not make it into BeeSecured Web due to time constraints, and they will be the first priority to future iterations of the project. The following features have been left out for future iterations:

**Table 7.1: Unimplemented Features**

Feature	Profile
Display sensor data history	User
Display alarm data history	User
Select a region on the map display present devices	User

Note: The last feature will require additional third party libraries as the capability is not included with Google Maps.

#### Notification Implementations

The current application cannot create additional sites from the user interface, so a new site must be created on the database. Future iterations should support creating additional sites as well as handling notifications from any changes to the site table. BeeSecured Web also currently relies on an internal checker function to ensure the consistency of displayed data to the database. In the future, it should be relying on notifications to asynchronously check and notify the user to refresh the application if changes have been made directly applied to the database.

## **8.2. Testing**

BeeSecured Web has not gone through a full cycle of feature testing or regression testing, and some effort will need to be allocated to this. Testing will also need to be performed under a WSN test site with physical devices operating in real time.

## **8.3. Map Upgrades**

### **8.3.1. *Google Maps v3***

BeeSecured Web currently uses Google Maps API version 2, which is the most recent version with the completed GWT library. Currently there is an open source project dedicated to wrapping Google Map's API version 3 to the GWT library, but many of the features are not complete and the documentation is also unfinished. Maps version 3 has many new features including custom animations, customizable marker information windows, deprecation of API keys that tie Google maps to a specific web site, and many more enhancements that would improve user experience with BeeSecured Web.

### **8.3.2. *GWT Google Map Utilities***

There are several open source JavaScript utilities written for Google Maps, and some of which have been wrapped as GWT libraries. Integrating these utilities can improve the overall performance of BeeSecured Web as well as add new capabilities that are not supported natively by the Google Maps API. Some useful capabilities include MarkerClusterer, which optimizes view by selectively displaying large number of markers by geographical regions, label markers, and context view control in Maps.

### **8.3.3. *Mapstraction***

Mapstraction is a JavaScript library that wraps all popular map services (Google Maps, Yahoo! Maps, MapQuest, etc) into one API, which allows developers eliminate the dependency of a particular map provider and easily change the service without rewriting any code. Currently there is no Mapstraction library written for the GWT framework, but if one is available in the future this would be a great direction to upgrade

BeeSecured Web. If the benefit of Mapstraction proves that it is worth the development cost, BeeSecured Web can potentially move in the direction of JavaScript development.

## **8.4. Bug Fixes and Code Refactor**

As with any software, the current version of BeeSecured Web is not bug free and these issues should be addressed in future iterations. Refactoring code to optimize performance of BeeSecured Web can improve the overall experience of the end user.



## **9. Conclusion**

Overall this project has been very successful, as all of the high priority requirements have been implemented into the application. The project timeline was fairly tight as many design changes were made after the initial research phase. Some of the requirements also came later on in the project, which caused some additional overhead later in the project. Although there were some delays with architecture changes and requirements, the technology learning phase was done in parallel, which allowed the first prototype to be presented in a reasonable amount of time.

Many decisions to simplify the overall architecture, including the decision to avoid firmware implementation on the Xport Pro, turned out favourably as the rest of the development effort was focused on the web application itself. The decision to utilize a local database was also a good one as it enabled smoother integration and testing between the BeeSecured Web and the BeeSecured Server applications.

## References

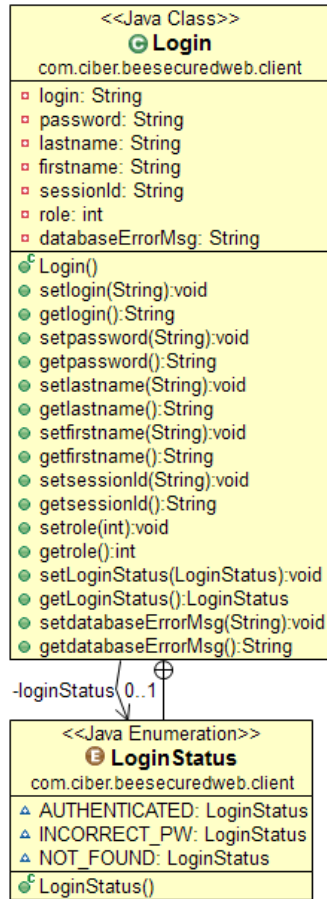
- Google. (2012). *Google Maps API Terms of Service*. Retrieved from [http://code.google.com/apis/maps/terms.html#section\\_4\\_4](http://code.google.com/apis/maps/terms.html#section_4_4)
- Yahoo. (2012). *Yahoo Maps Terms of Use*. Retrieved from <http://info.yahoo.com/legal/us/yahoo/maps/mapsapi/mapsapi-2141.html>
- Microsoft. (2012). *Bing Maps Licensing and Pricing Information*. Retrieved from <http://www.microsoft.com/maps/product/licensing.aspx>
- Google. (2012). *Google Web Toolkit Documentation*. Retrieved from <http://code.google.com/webtoolkit/overview.html>
- Google. (2012). *Google App Engine Documentation*. Retrieved from <http://code.google.com/appengine/docs/>
- Amazon. (2012). *Amazon Web Services*. Retrieved from <http://aws.amazon.com/ec2/>
- Microsoft. (2012). *Microsoft Azure*. Retrieved from <http://www.windowsazure.com/en-us/>
- Lantronix. (2011). *Lantronix Xport Pro*. Retrieved from <http://www.lantronix.com/device-networking/embedded-device-servers/xport-pro.html>
- Apache. (2012). *Apache Tomcat*. Retrieved from <http://tomcat.apache.org/>
- BST. (2012). *BeeSecured Database Document*. Retrieved from [http://ciber-linux1.ensc.sfu.ca/redmine/projects/beesebsite/wiki/Database\\_Description](http://ciber-linux1.ensc.sfu.ca/redmine/projects/beesebsite/wiki/Database_Description)

## **Appendices**

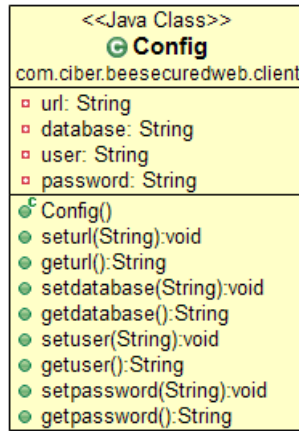
## Appendix A.

### UML for Client Application

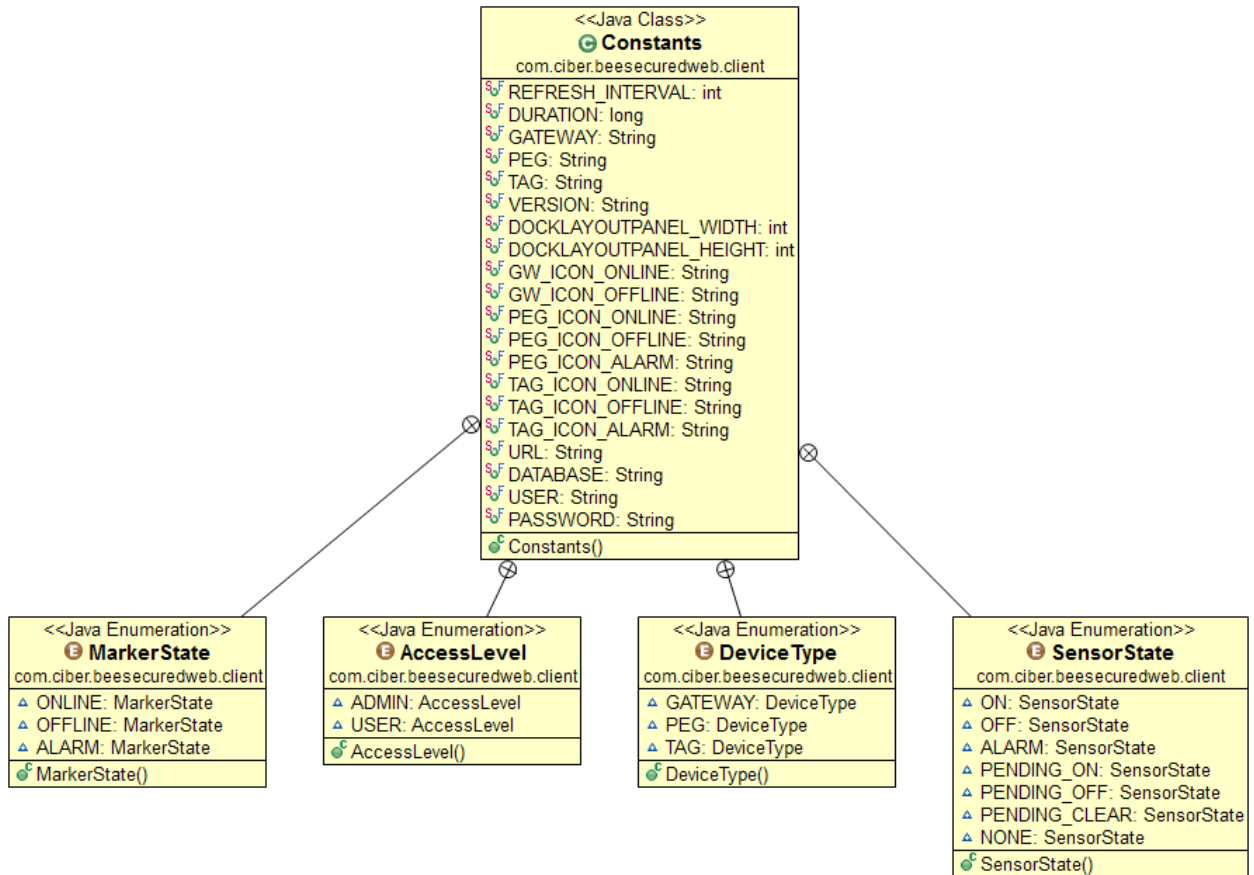
Figure A.1: Login Class



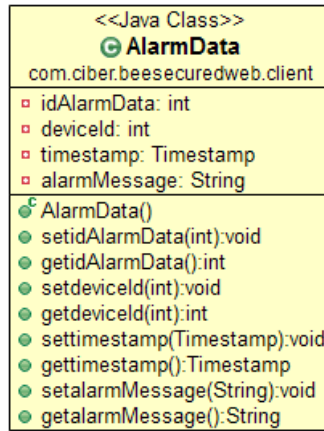
**Figure A.2: Config Class**



**Figure A.3: Constants Class**



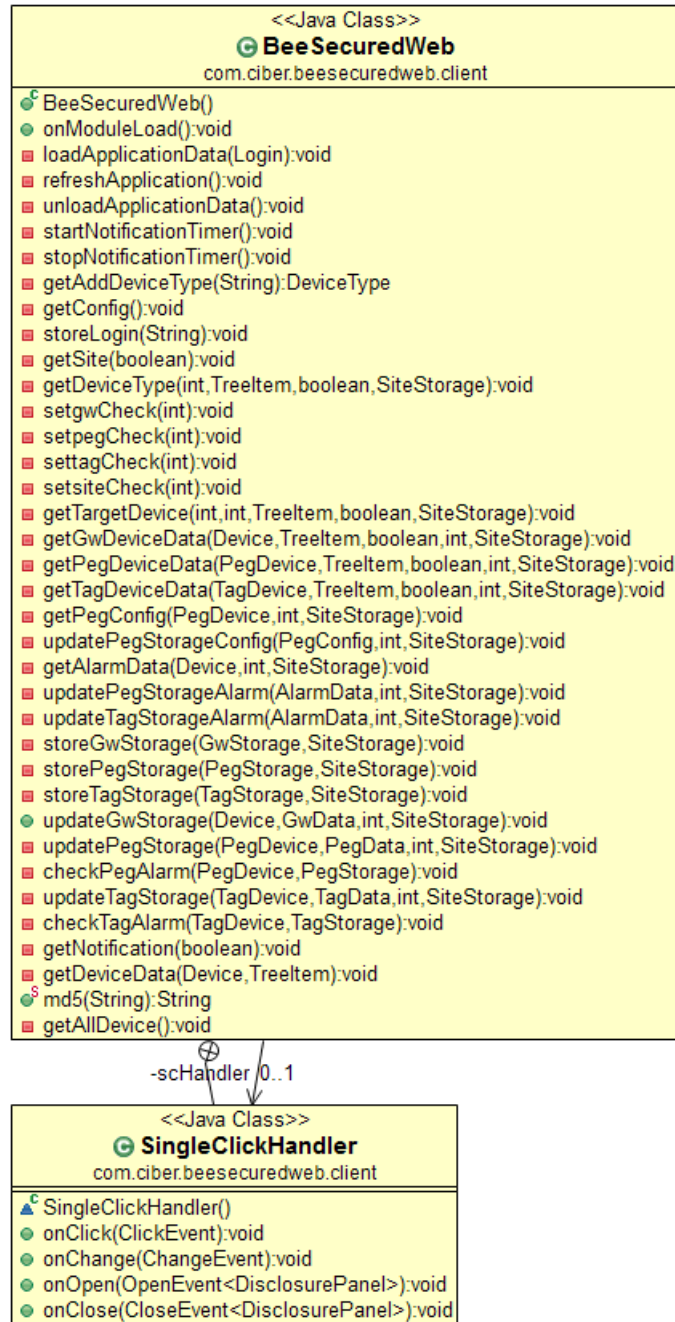
**Figure A.4: AlarmData Class**



**Figure A.5: BeeSecuredWeb Class (Fields)**

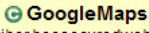
<<Java Class>>  <b>BeeSecuredWeb</b> com.ciber.beeseuredweb.client	
inputdb_btn:	Button
query_btn:	Button
alarm_disclosurePanel:	DisclosurePanel
deviceExplorer_disclosurePanel:	DisclosurePanel
rootPanel:	RootPanel
output_textbox:	TextBox
query_textbox:	TextBox
map:	MapWidget
test_disclosurePanel:	DisclosurePanel
googleMaps:	GoogleMaps
peg_flexTable:	FlexTable
peg_treeltem:	Treeltem
tag_treeltem:	Treeltem
gateway_treeltem:	Treeltem
scHandler:	SingleClickHandler
pegSetting:	Button
tag_flexTable:	FlexTable
test_alarm_button:	Button
dockLayoutPanel:	DockLayoutPanel
deviceExplorer_tree:	Tree
test_toggleButton:	ToggleButton
alarms_flexTable:	FlexTable
alarms_scrollPanel:	ScrollPanel
refreshDialogBox:	DialogBox
refreshDbButton:	Button
refreshDbLabel:	Label
refreshDbvp:	VerticalPanel
gwCheck:	int
pegCheck:	int
tagCheck:	int
siteCheck:	int
siteList:	ArrayList<SiteStorage>
deviceEdit_toggleButton:	ToggleButton
deviceEdit_listBox:	ListBox
lblDeviceToBe:	Label
newDeviceType:	DeviceType
deviceEdit_verticalPanel:	VerticalPanel
deviceEdit_horizontalPanel:	HorizontalPanel
deviceEdit_applyButton:	Button
deviceEdit_cancelButton:	Button
refreshTimer:	Timer
loginPopup:	DecoratedPopupPanel
userTextBox:	TextBox
passwordTextBox:	PasswordTextBox
remembermeCheckBox:	CheckBox
loginButton:	Button
loginMsgLabel:	Label
login_anchor:	Anchor
username_label:	Label
deviceEdit_disclosurePanel:	DisclosurePanel
version_label:	Label
beeseured_logo:	Image
north_userHorizontalPanel:	HorizontalPanel
north_loginHorizontalPanel:	HorizontalPanel
absolutePanel:	AbsolutePanel
dockLayoutPanelWidth:	int
dockLayoutPanelHeight:	int
north_titleHorizontalPanel:	HorizontalPanel

**Figure A.6: BeeSecuredWeb Class (Methods)**





**Figure A.7: GoogleMaps Class (Fields)**

<<Java Class>>  <b>GoogleMaps</b> com.ciber.beesecuredweb.client	
map	MapWidget
LatLngList	ArrayList<LatLng>
MarkerList	ArrayList<Marker>
markernum	int
map_options	MapOptions
pegConfigTable	FlexTable
gwConfigTable	FlexTable
tagConfigTable	FlexTable
pegSignalTable	FlexTable
extSensorTypeList	ArrayList<Integer>
pegSignalDialogBox	DialogBox
settingsDialog	DialogBox
gwSettingsDialog	DialogBox
pegSettingsDialog	DialogBox
tagSettingsDialog	DialogBox
signalAbutton	Button
signalBbutton	Button
signalCbutton	Button
signalDbutton	Button
sandler	SingleClickHandler
checkBoxArray	CheckBox[]
pegSignalAvalue	long
pegSignalBvalue	long
pegSignalCvalue	long
pegSignalDvalue	long
activePegSignalDialog	char
signalBitArray	int[]
mscHandler	MapSingleClickHandler
pegInfoTable	FlexTable
pegDeviceIdLabel	Label
pegTemperatureLabel	Label
pegTimestampLabel	Label
pegBatteryLabel	Label
pegSensorAAlarmLabel	Label
pegSensorBAlarmLabel	Label
pegVibAlarmLabel	Label
pegTempAlarmLabel	Label
pegAlarmTimeLabel	Label
gwInfoTable	FlexTable
tagInfoTable	FlexTable
tagDeviceIdLabel	Label
tagHeartRateLabel	Label
tagTimestampLabel	Label
tagBodyAttitudeLabel	Label
tagHeartRateAlarmLabel	Label
tagFallenAlarmLabel	Label
tagAlarmTimeLabel	Label
sensorConfigDialog	DialogBox
sensorConfigTable	FlexTable
sensorAcheckbox	CheckBox
sensorBcheckbox	CheckBox
vsensorcheckbox	CheckBox
tsensorcheckbox	CheckBox
cancelSensorConfigBtn	Button
tagSensorConfigDialog	DialogBox
tagSensorConfigTable	FlexTable
sensorHcheckbox	CheckBox
sensorFcheckbox	CheckBox
tagCancelSensorConfigBtn	Button
clearAlarmDialog	DialogBox
clearAlarmVp	VerticalPanel
cancelClearAlarmBtn	Button
extSensorTypeList	ArrayList<ExtSensorType>
pegStorageList	ArrayList<PegStorage>
gwStorageList	ArrayList<GwStorage>
tagStorageList	ArrayList<TagStorage>
showDetails	boolean
editDeviceList	ArrayList<Device>
editMarkerList	ArrayList<Marker>
mapStartZoom	int
mapStartCoord	LatLng
accessLevel	AccessLevel

**Figure A.8: GoogleMaps Class (Methods)**

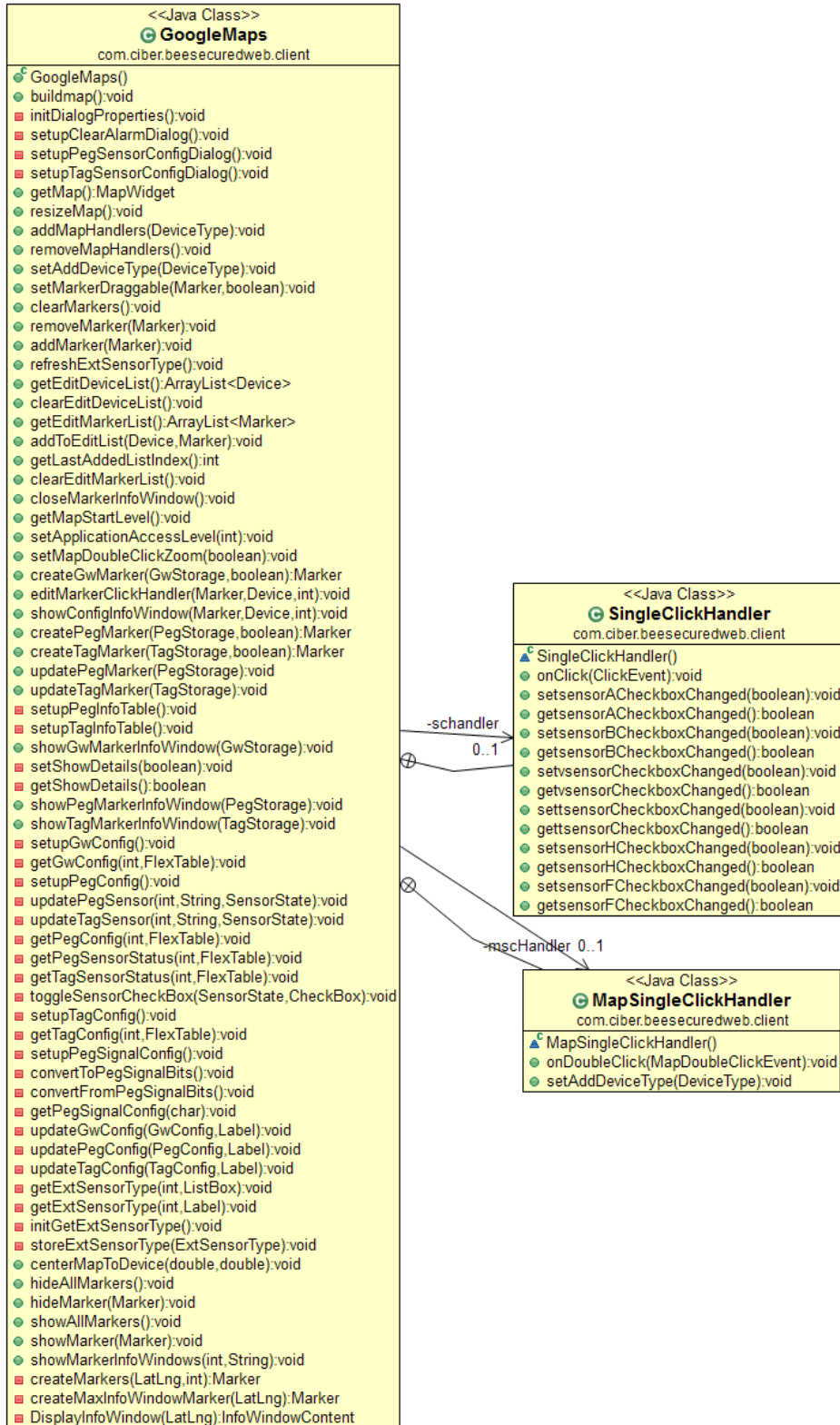
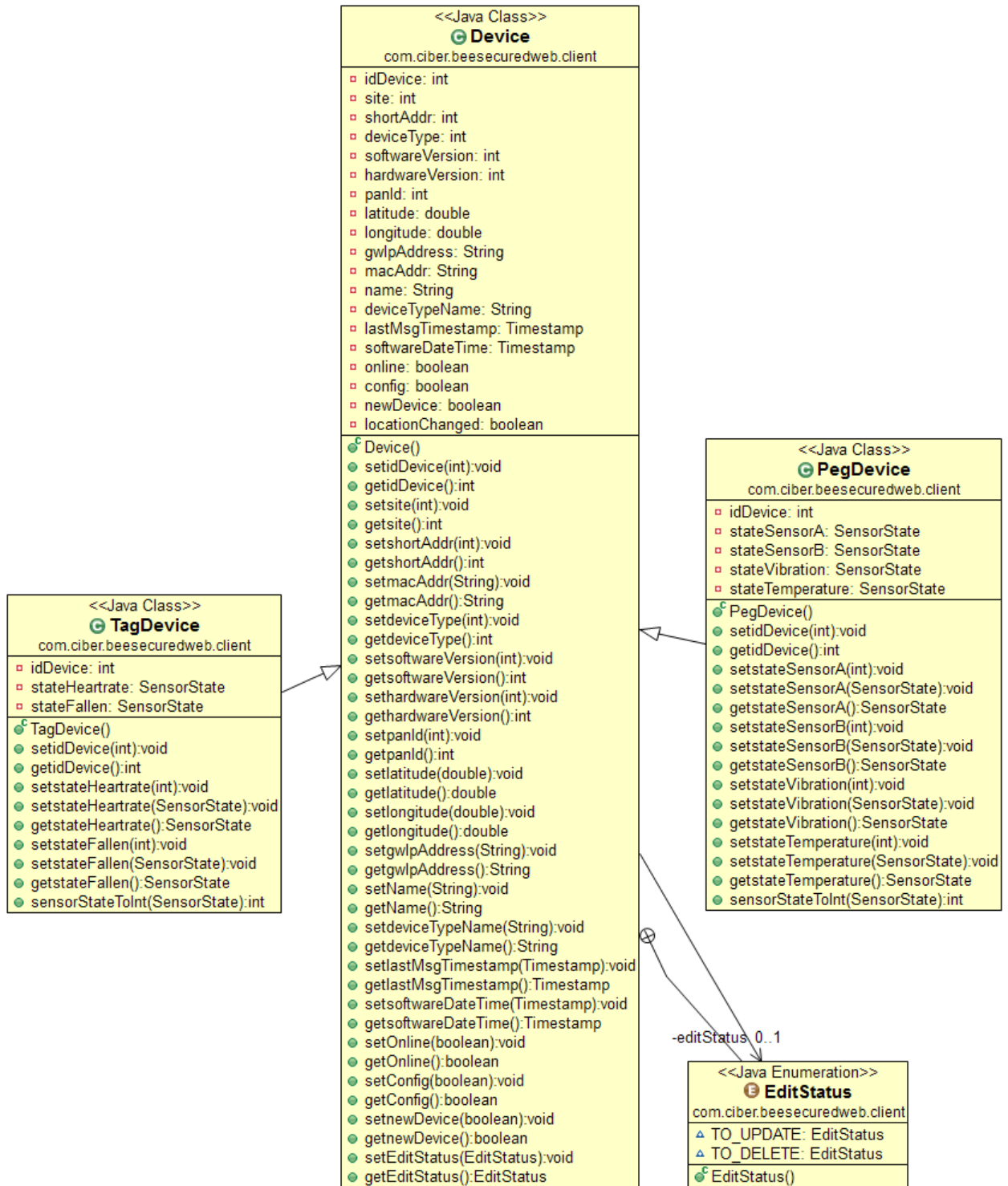


Figure A.9: Device, TagDevice, PegDevice Classes



**Figure A.10: DeviceService, DeviceServiceAsync Class**

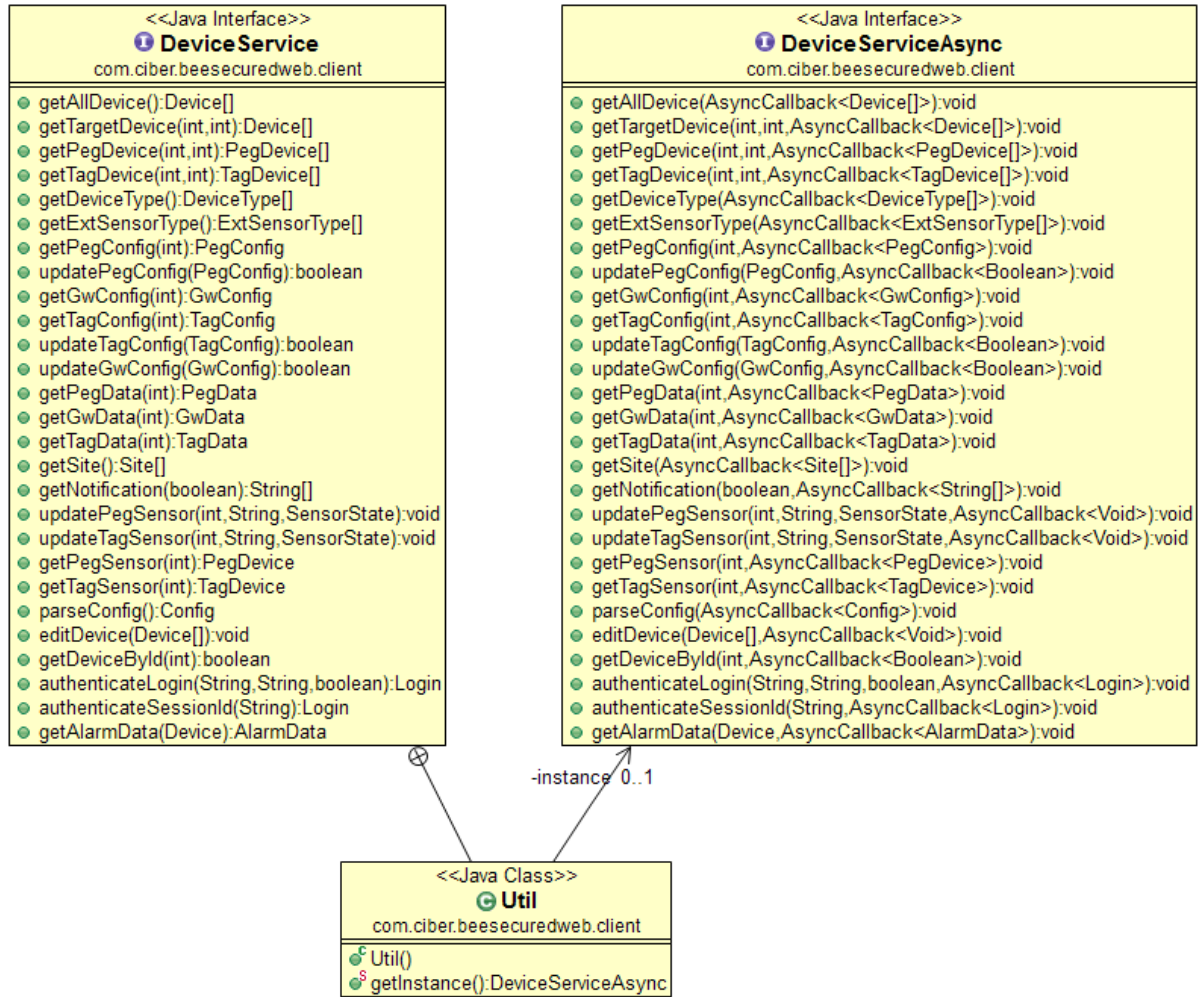


Figure A.11: DeviceType, ExtSensorType

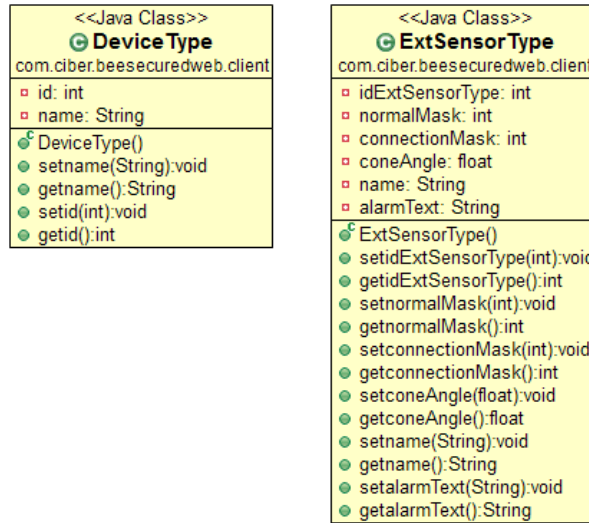


Figure A.12: SiteStorage, GwStorage, PegStorage, TagStorage, Site Classes

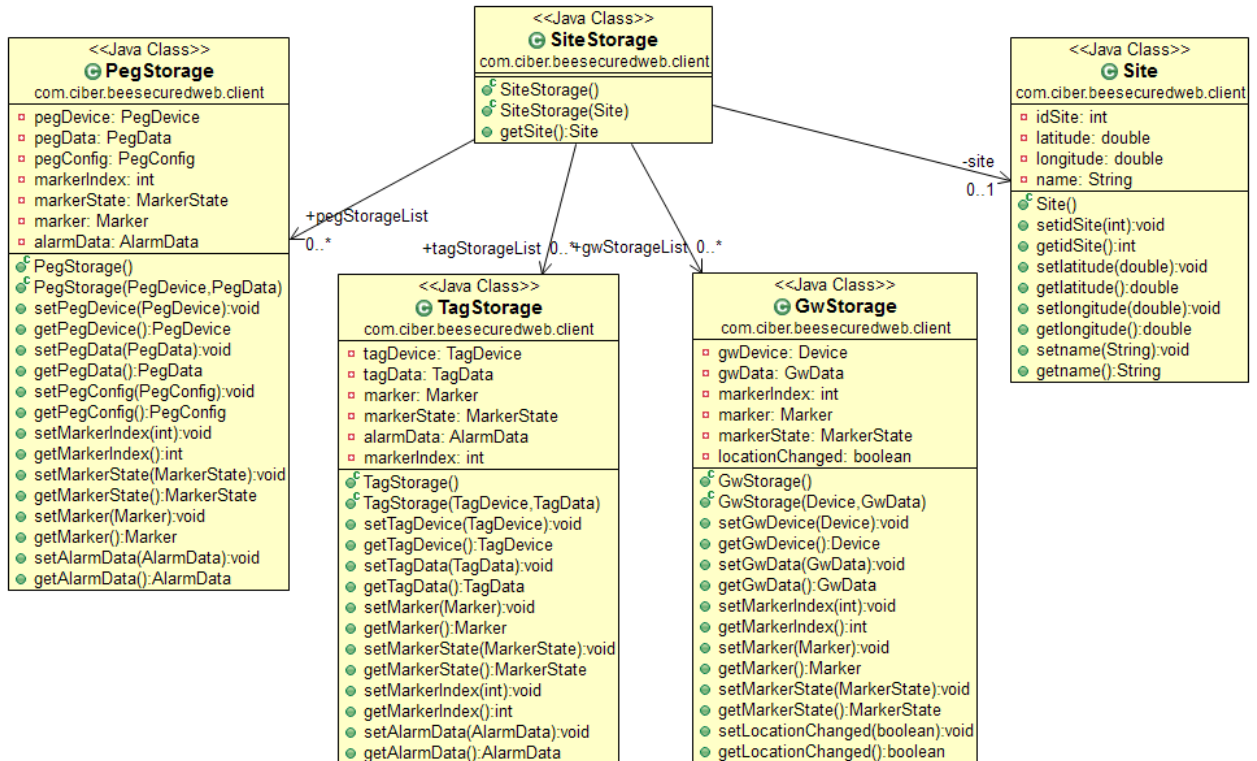


Figure A.13: GwStorage Class

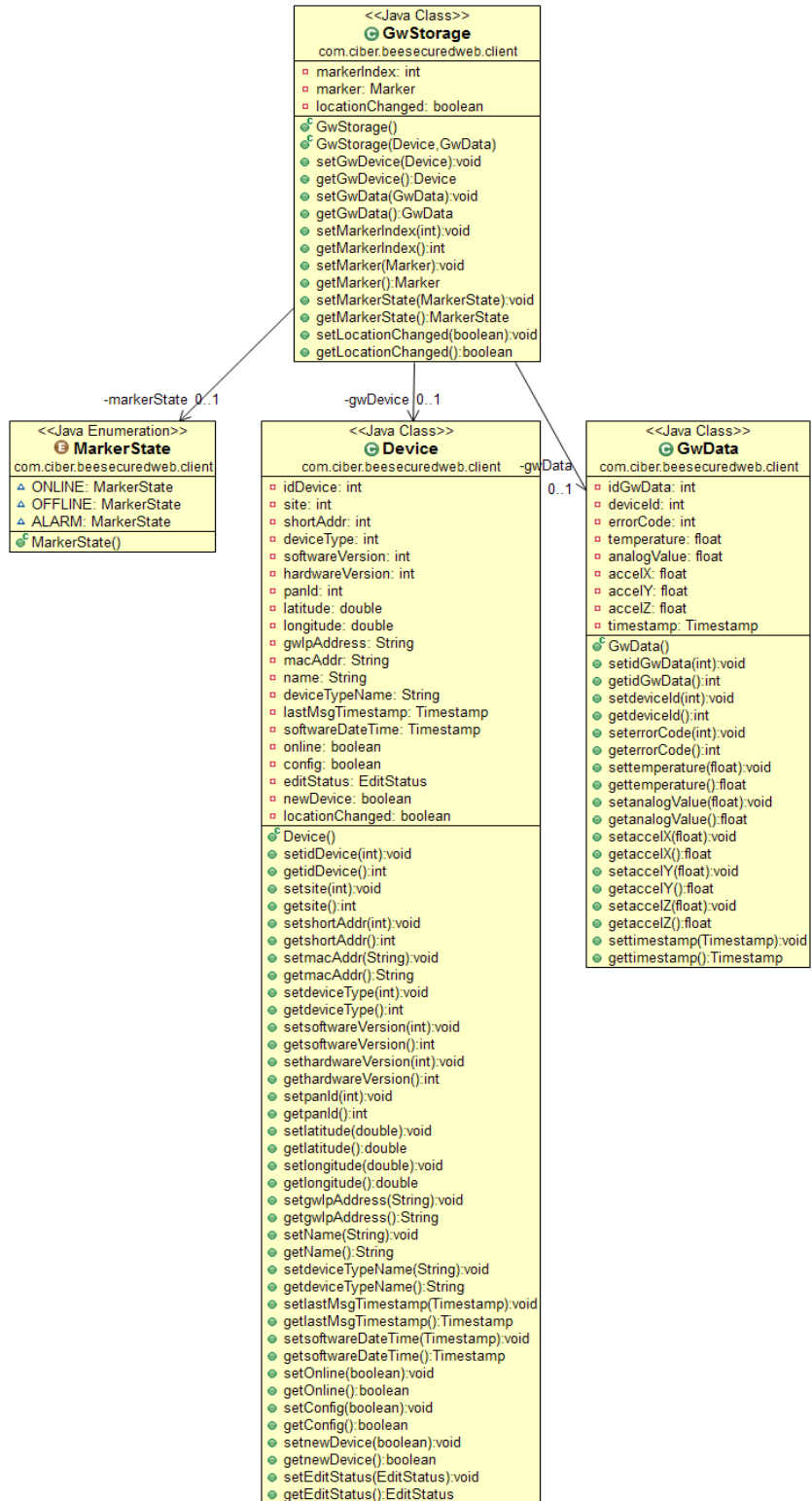




Figure A.14: PegStorage Class

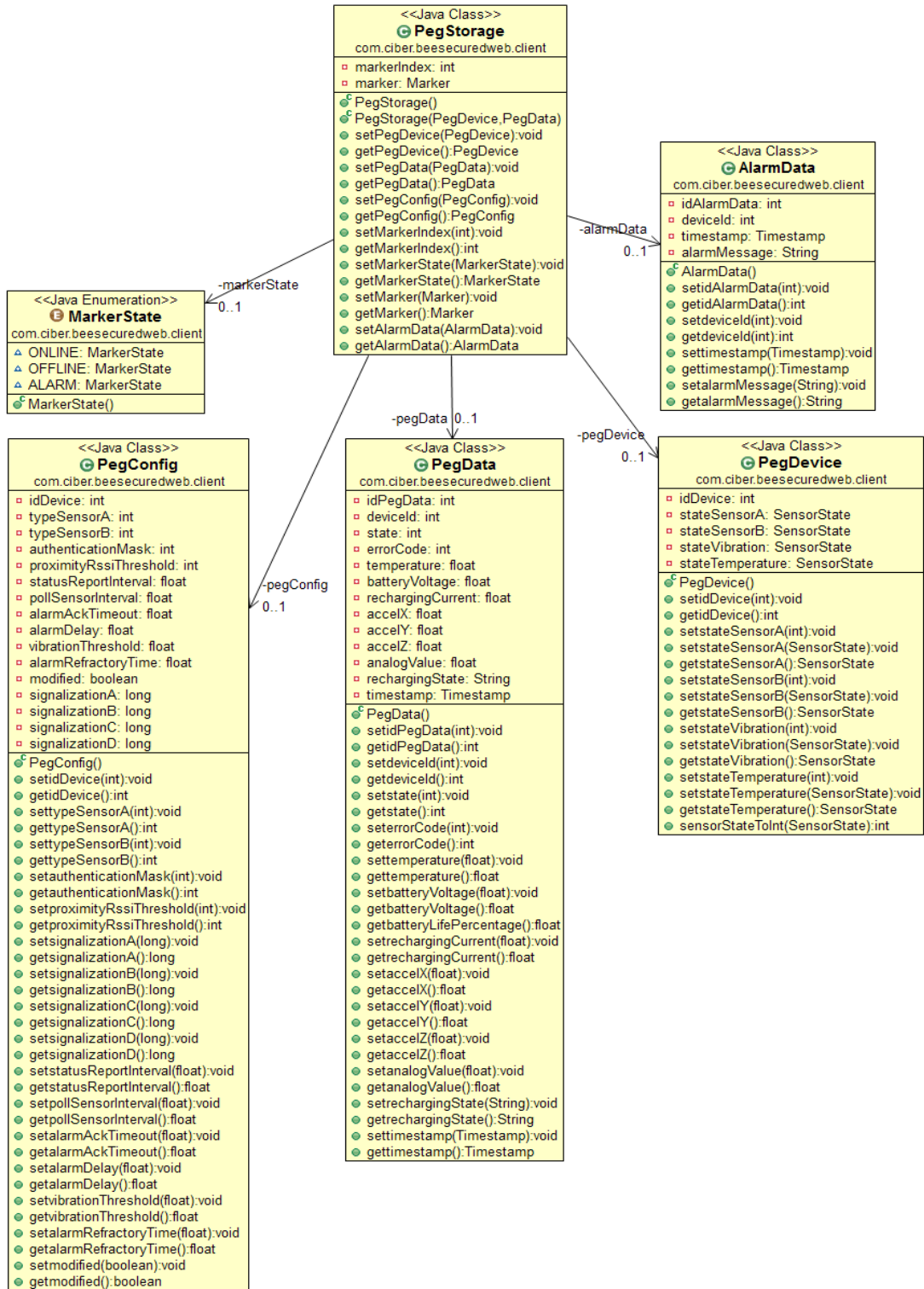
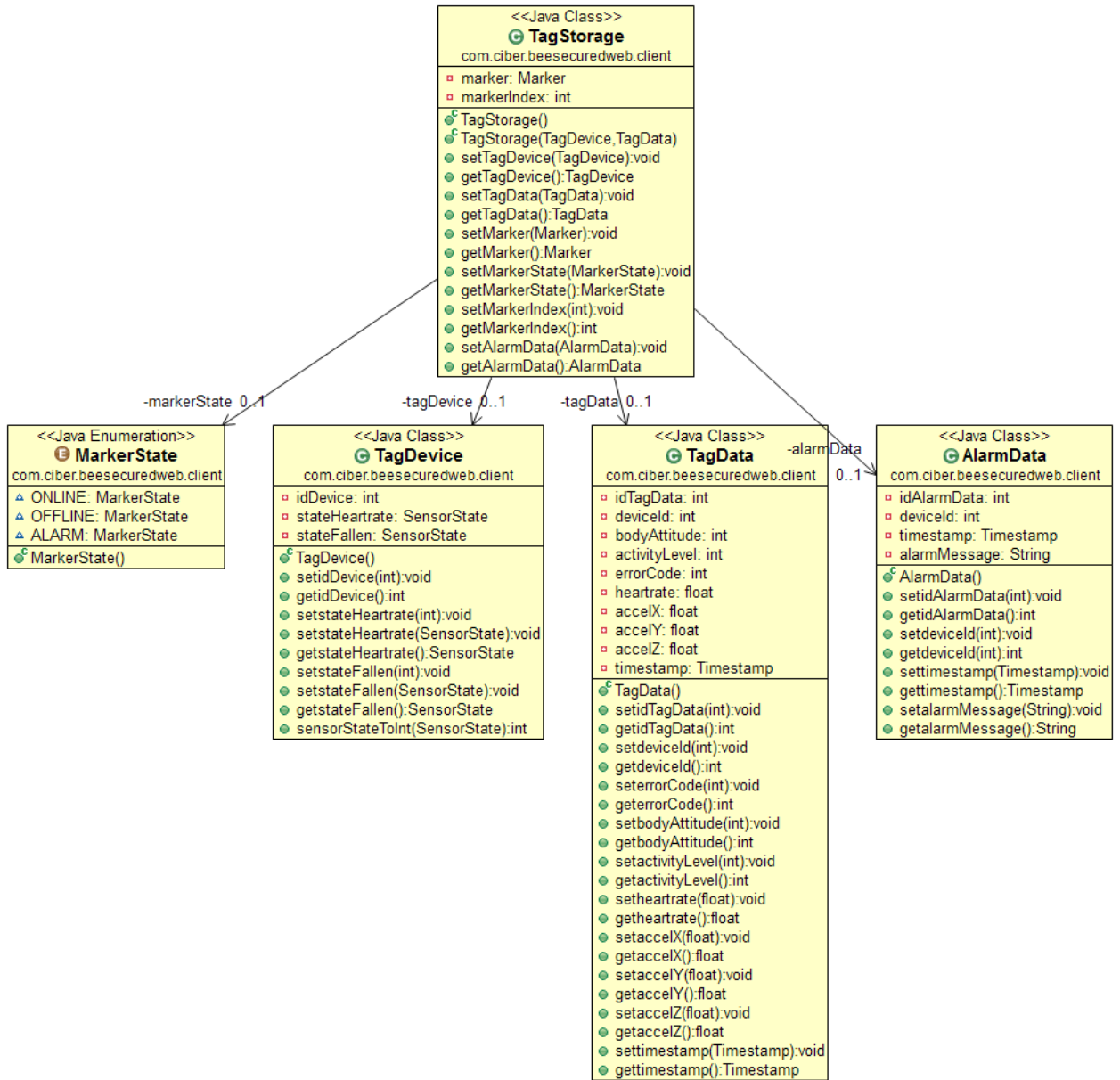


Figure A.15: TagStorage Class





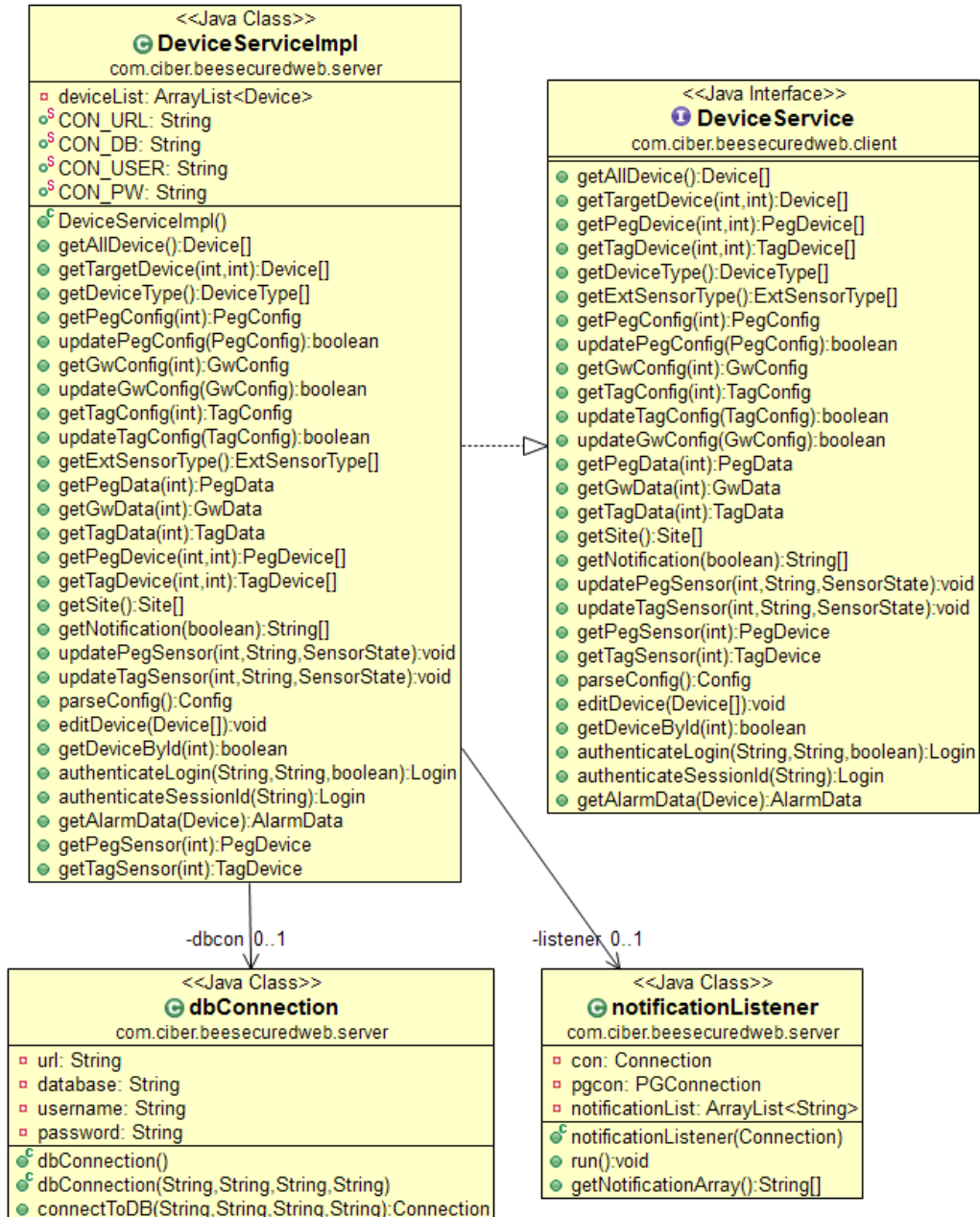
**Figure A.16: GwConfig, PegConfig, TagConfig Classes**

<pre> &lt;&lt;Java Class&gt;&gt; <b>PegConfig</b> com.ciber.beesecuredweb.client  idDevice: int typeSensorA: int typeSensorB: int authenticationMask: int proximityRssiThreshold: int statusReportInterval: float pollSensorInterval: float alarmAckTimeout: float alarmDelay: float vibrationThreshold: float alarmRefractoryTime: float modified: boolean signalizationA: long signalizationB: long signalizationC: long signalizationD: long  PegConfig() setidDevice(int):void getidDevice():int settypeSensorA(int):void gettypeSensorA():int settypeSensorB(int):void gettypeSensorB():int setauthenticationMask(int):void getauthenticationMask():int setproximityRssiThreshold(int):void getproximityRssiThreshold():int setsignalizationA(long):void getsignalizationA():long setsignalizationB(long):void getsignalizationB():long setsignalizationC(long):void getsignalizationC():long setsignalizationD(long):void getsignalizationD():long setstatusReportInterval(float):void getstatusReportInterval():float setpollSensorInterval(float):void getpollSensorInterval():float setalarmAckTimeout(float):void getalarmAckTimeout():float setalarmDelay(float):void getalarmDelay():float setvibrationThreshold(float):void getvibrationThreshold():float setalarmRefractoryTime(float):void getalarmRefractoryTime():float setmodified(boolean):void getmodified():boolean </pre>	<pre> &lt;&lt;Java Class&gt;&gt; <b>GwConfig</b> com.ciber.beesecuredweb.client  idDevice: int statusReportInterval: int routeDiscoveryInterval: float modified: boolean  GwConfig() setidDevice(int):void getidDevice():int setstatusReportInterval(int):void getstatusReportInterval():int setrouteDiscoveryInterval(float):void getrouteDiscoveryInterval():float setmodified(boolean):void getmodified():boolean </pre>	<pre> &lt;&lt;Java Class&gt;&gt; <b>TagConfig</b> com.ciber.beesecuredweb.client  idDevice: int authenticationMask: int statusReportInterval: float pollSensorInterval: float alarmAckTimeout: float nearbyBeaconInterval: float alarmRefractoryTime: float fallenTime: float fallenThreshold: float activityThreshold: float inactivityThreshold: float modified: boolean  TagConfig() setidDevice(int):void getidDevice():int setauthenticationMask(int):void getauthenticationMask():int setstatusReportInterval(float):void getstatusReportInterval():float setpollSensorInterval(float):void getpollSensorInterval():float setalarmAckTimeout(float):void getalarmAckTimeout():float setnearbyBeaconInterval(float):void getnearbyBeaconInterval():float setalarmRefractoryTime(float):void getalarmRefractoryTime():float setfallenTime(float):void getfallenTime():float setfallenThreshold(float):void getfallenThreshold():float setactivityThreshold(float):void getactivityThreshold():float setinactivityThreshold(float):void getinactivityThreshold():float setmodified(boolean):void getmodified():boolean </pre>
---	---	--

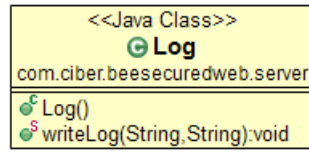
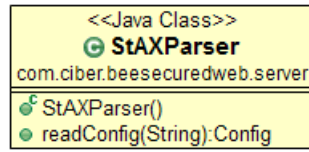
## Appendix B.

### UML for Server Application

Figure B.1: Servlet Classes



**Figure B.2: Servlet Classes Continued**



## Appendix C.

### Database Tables

**Table C.1: Site Table**

Column	Data type	Description	Range
idSite	int	Primary key	
name	string	Name of the network/site	
latitude	double	Position [deg]	
longitude	double	Position [deg]	

Note. BST (2011)

**Table C.2: Device Table**

Column	Data type	Description	Range
idDevice	int	Primary key	
name	string	Name of device	
site	int	Ref. to the site of this device	
latitude	double	Position [deg]	
longitude	double	Position [deg]	
shortAddr	int	Zigbee short address	
macAddr	string	MAC/IEEE address of device	
deviceType	int	type of device (see deviceType table)	
gwIpAddress	string	IP address of gateway to device	
lastMsgTimestamp	dateTime	time of last received message	
softwareVersion	int Version	number of the firmware	[0, 255]
softwareDateTime	dateTime	Build time and date of the software	
hardwareVersion	int	Version number of the hardware	[0, 255]
panId	int	PAN Id of the Zigbee network 16 bit	
online	boolean	Flags if the device is online or not	

Note. BST (2011)

**Table C.3: Peg Config Table**

Column	Data type	Description	Range
idDevice	int	Primary key	
typeSensorA	int	Ref. to the type of external sensor A	
typeSensorB	int	Ref. to the type of external sensor B	
statusReportInterval	float	Interval at which status reports are send [s]	[0.0, 65535.0]
pollSensorInterval	float	Interval at which to poll sensors [s]	[0.0, 65.0]
alarmAckTimeout	float	Timeout for alarm acknowledgements [s]	[0, 65.0]
alarmDelay	float	Delay before alarms are send, to give TAGs time to disable alarms [s]	[0, 65.0]
vibrationThreshold	float	Acceleration threshold to detect vibrations [m/s <sup>2</sup> ]	[0.0, 4.0]
alarmRefractoryTime	int	Time between raising an alarm condition and raising the same condition again [s]	[0, 255]
authenticationMask	int	Mask to authenticate TAGs to disable alarms for smart intrusion 16 bit	
proximityRssiThreshold	int	RSSI values to detect TAG proximity for smart intrusion	[-128, 0]
lowerTempThreshold	float	Lower threshold for temperature alarm	[-40.0, 125.0]
upperTempThreshold	float	Upper threshold for temperature alarm	[-40.0, 125.0]
signalizationA	int	Mask for mapping alarms to signalization channel A	
signalizationB	int	Mask for mapping alarms to signalization channel B	
signalizationC	int	Mask for mapping alarms to signalization channel C	
signalizationD	int	Mask for mapping alarms to signalization channel D	
modified	bool	True if modified by GUI (GUIs are only allowed to modify a row if false) False if confirmed by device	

*Note. BST (2011)*

**Table C.4: Tag Config Table**

Column	Data type	Description	Range
idDevice	int	Primary key	
statusReportInterval	float	Interval at which status reports are send [s]	[0.0, 65535.0]
pollSensorInterval	float	Interval at which to poll sensors [s]	[0.0, 65535.0]
alarmAckTimeout	int	Timeout for alarm acknowledgements [s]	[0, 255]
nearbyBeaconInterval	float	Interval at which to send "Hello" beacons [s]	
alarmRefractoryTime	int	Time between raising an alarm condition and raising the same condition again [s]	[0, 255]
fallenTime	float	Duration all axis have to be below fallenThreshold to trigger fallen alarm [s]	[0.0, 65535.0]
fallenThreshold	float	Acceleration threshold, all axis below this threshold trigger a fallen alarm [m/s <sup>2</sup> ]	[0.0, 65535.0]
activityThreshold	float	Accelerations higher trigger high activity [m/s <sup>2</sup> ]	
inactivityThreshold	float	Acceleration lower trigger low activity[m/s <sup>2</sup> ]	
inactivityTime	float	Time window to detect inactivity [s]	
authenticationMask	int	Mask to disable PEG alarms (smart intrusion)	
modified	bool	True if modified by GUI (GUIs are only allowed to modify a row if false) False if confirmed by device	

*Note. BST (2011)*

**Table C.5: Gateway Config Table**

Column	Data type	Description	Range
idDevice	int	Primary key	
statusReportInterval	int	Interval at which status reports are send [s]	[0, 65535]
routeDiscoveryInterval	int	Interval in which to send many to one route discoveries [s]	[0, 65535]
lowerTempThreshold	float	Lower threshold for temperature alarm	[-40.0, 125.0]
upperTempThreshold	float	Upper threshold for temperature alarm	[-40.0, 125.0]
modified	bool	True if modified by GUI (GUIs are only allowed to modify a row if false) False if confirmed by device	

*Note. BST (2011)*

**Table C.6: Alarm Data Table**

Column	Data type	Description	Range
idAlarmData	int	Primary key	
deviceId	int	Ref. to the device that generated the alarm	
timestamp	dateTime	Time of alarm	
alarmMessage	string	Text describing the alarm	
alarmVector	int	alarm code as send by device	

*Note. BST (2011)*

**Table C.7: External Sensor Type Table**

Column	Data type	Description	Range
idExtSensorType	int	Primary key	
name	string	name of sensor	
normalMask	int	GPIO mask for normal condition (any deviation causes an alarm)	
conectionMask	int	Masks used GPIO lines	
alarmText	string	Text describing the alarm	
coneAngle	float	Angle of the sensor cone [rad]	
bearing	float	Bearing of the sensor [rad]	
range	float	Range of the sensor [m]	

*Note.* BST (2011)

**Table C.8: Gateway Data Table**

Column	Data type	Description	Range
idGwdata	int	Primary key	
deviceId	int	Ref. to the device that generated the data	
timestamp	dateTime	Time stamp of the data record	
temperature	float	[deg C]	
aanlogValue	float	reading from the analog input	
accelX	float	Acceleration in X axis [m/s <sup>2</sup> ]	
accelY	float	Acceleration in Y axis [m/s <sup>2</sup> ]	
accelZ	float	Acceleration in Z axis [m/s <sup>2</sup> ]	
errorCode	int	Errors of the device	

*Note.* BST (2011)



**Table C.9: Peg Data Table**

Column	Data type	Description	Range
idPegdata	int	Primary key	
deviceId	int	Ref. to the device that generated the data	
timestamp	dateTime	Time stamp of the data record	
temperature	float	[deg C]	
batteryVoltage	float	[V]	
rechargingCurrent	float	[A]	
rechargingState	string	?	
accelX	float	Acceleration in X axis [m/s <sup>2</sup> ]	
accelY	float	Acceleration in Y axis [m/s <sup>2</sup> ]	
accelZ	float	Acceleration in Z axis [m/s <sup>2</sup> ]	
errorCode	int	Errors of the device	
state	int	State of the device	
analogValue	float	Reading of the analog input	

Note. BST (2011)

**Table C.10: Tag Data Table**

Column	Data type	Description	Range
idTagdata	int	Primary key	
deviceId	int	Ref. to the device that generated the data	
timestamp	dateTime	Time stamp of the data record	
heartrate	float	Heart rate [bpm]	
accelX	float	Acceleration in X axis [m/s <sup>2</sup> ]	
accelY	float	Acceleration in Y axis [m/s <sup>2</sup> ]	
accelZ	float	Acceleration in Z axis [m/s <sup>2</sup> ]	
bodyAttitude	int	Attitude of the person wearing the tag	
activityLevel	int	Activity level of the person wearing the tag	
errorCode	int	Errors of the device	

Note. BST (2011)

**Table C.11: Tag Sensor Table**

Column	Data type	Description	Range
idDevice	int	Primary key	
stateHeartrate	int	State of the heart rate sensor (see sensorState)	
stateFallen	int	State of the fallen sensor (see sensorState)	

Note. BST (2011)

**Table C.12: Device Table**

Column	Data type	Description	Range
idDevice	int	Primary key	
stateSensorA	int	State of the external sensor A (see sensorState)	
stateSensorB	int	State of the external sensor B (see sensorState)	
stateVibration	int	State of the vibration sensor (see sensorState)	
stateTemperature	int	State of the temperature sensor (see sensorState)	

Note. BST (2011)

**Table C.13: User Role Table**

Column	Data type	Description	Range
idUserRole	int	Primary key	
name	varchar(45)	Name of role	

Note. BST (2011)

**Table C.14: Login Table**

Column	Data type	Description	Range
login	varchar(45)	Primary key, login name	
role	int	Reference to the userRole table	
password	char(32)	MD5 hash of the password	
lastname	varchar(45)	last name of the user	
firstname	varchar(45)	first name of the user	
sessionid	varchar(45)	sessionid of user connection	

Note. BST (2011)

## Fixed Value Tables

**Table C.15: Device Type Table**

idDeviceType	name
1	GW
2	PEG
3	TAG

Note. BST (2011)

**Table C.16: Sensor State Table**

idSensorState	state
1	On
2	Off
3	Alarm
4	Pending_On
5	Pending_Off
6	Pending_Clear

Note. BST (2011)