TED AggreData
Software Architecture Document

Version 1.00

Revision History

Date	Version	Description	Author
06/01/2011	1.00	Initial Version	Pete Arvanitis

Table of Contents

1. Introduction

TED AggreData, developed for Center Street Technologies, is a web application designed to facilitate scheduling of information sharing events as they relate to a legal action. Its intent is to serve as a 'common middle' for heterogeneous scheduling systems used by the various stakeholders (firms, lawyers, consultants and clients) involved in the action.

1.1. Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2. Scope

This document provides the architectural overview of the TED AggreData application.

1.3. Definitions, Acronyms, and Abbreviations

- **Registered User –** a system role. A registered user is able to log into the AggreData system to view their data and administer their account
- **Administrator –** a system role. An administrator is an extension of a registered user. In addition to administer their own account, the administrator can administer all other user accounts.
- **TED 5000 "Gateway"** – Also referred to as just a **TED 5000.** These are the TED 5000 products sold by The Energy Detective. They have a 3rd party post option that will allow energy data to be posted to the AggreData Server
- **Location-** The location of one or more TED 5000's. A TED 5000 can only belong to one location.
- **Group -** A grouping of one or more locations. Locations can belong to more than one group.

1.4. References

- *TED 5000 3rd Party Post API*
- *TED 5000 API*

1.5. Overview

The main organization of the document is based on RUP's 4+1 Architectural Views. Each view describes the TED AggreData implementation and its impact on the overall architecture.

2. Architectural Representation

The TED AggreData web application architecture is based on a multi-layered architecture written in as a Java Web Application using the Google Web Toolkit to provide the front-end.

- Use-Case View – This section outlines the Requirements by referencing the significant use cases of the system and mapping them to the corresponding use case realizations.
- Logical View – This section describes the architecturally significant parts of the design model such as its decomposition into subsystems and packages. And for each significant package, its decomposition into classes and class utilities.
- Process View – This view is used to describe the various processes (single threads of control) and process groups that make up the system.
- Deployment View - This section explains the physical (hardware) configurations on which the software is deployed and run
- Implementation View – This section describes the overall structure of the implementation model, the decomposition of the software into layers and subsystems in the implementation model, and any architecturally significant components.

3. Architectural Goals and Constraints

3.1. Security

- Security will be managed by the application itself
- Access to workflows will be based on user roles.

3.2. Use of Off-the-Self Products

- The system will be a web application deployed to java servlet container. For development purposes we will be developing against a Jetty server
- The database will be MySQL 5.x
- The Google Web Toolkit (GWT) will be used for building the user interface
- Bone Connection Pool will be used for JDBC management

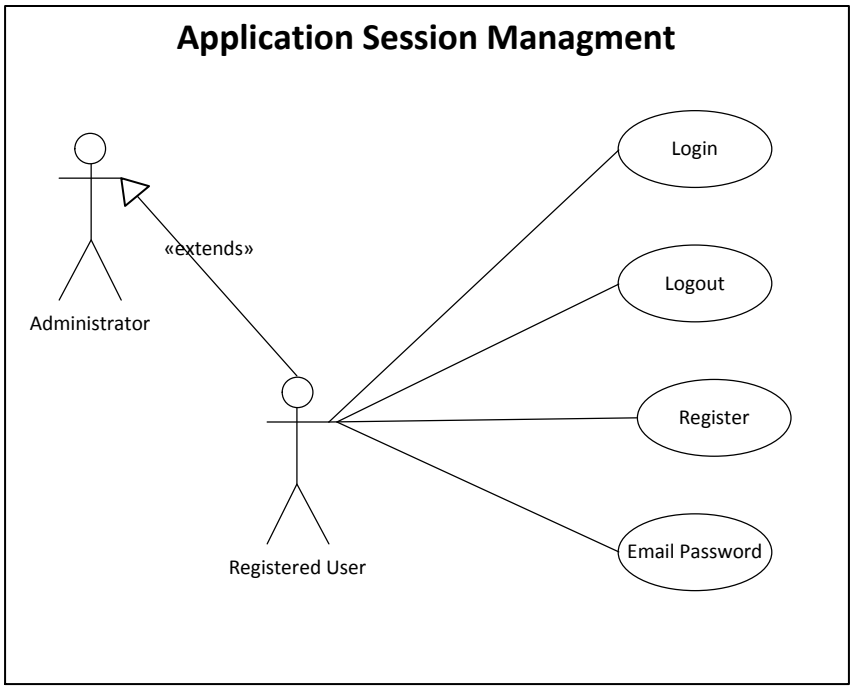
3.3. External Integration

- TED5000 Units in the field will post data to TED AggreData using their built-in 3rd Party Post Feature (including Activation)
- Third party applications will communicate with the TED AggreData server using a REST API
- TED AggreData will poll the National Weather Service for Weather Information

4. Use-Case View

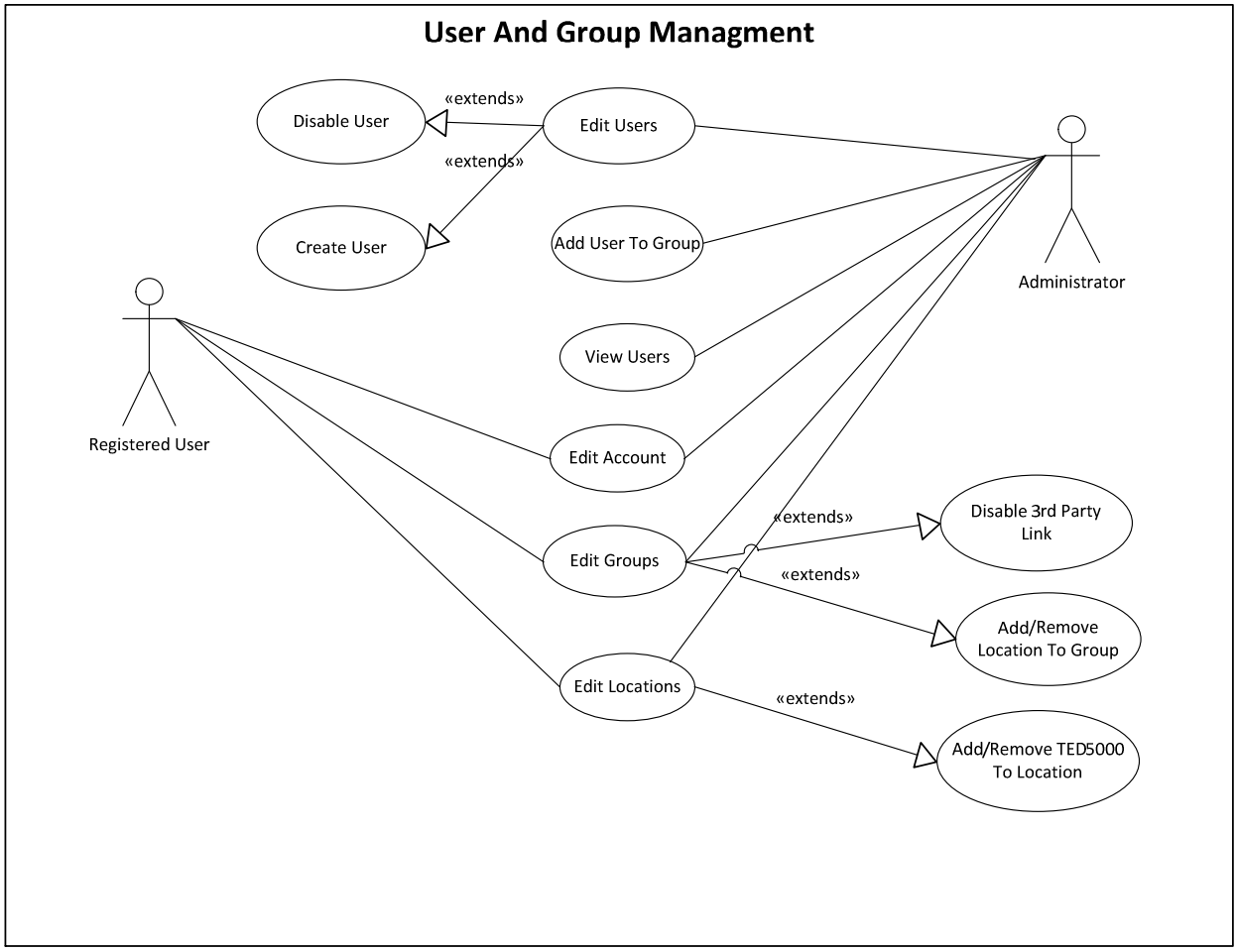
4.1. High Level Use Case Diagrams

4.1.1.Application Session Management



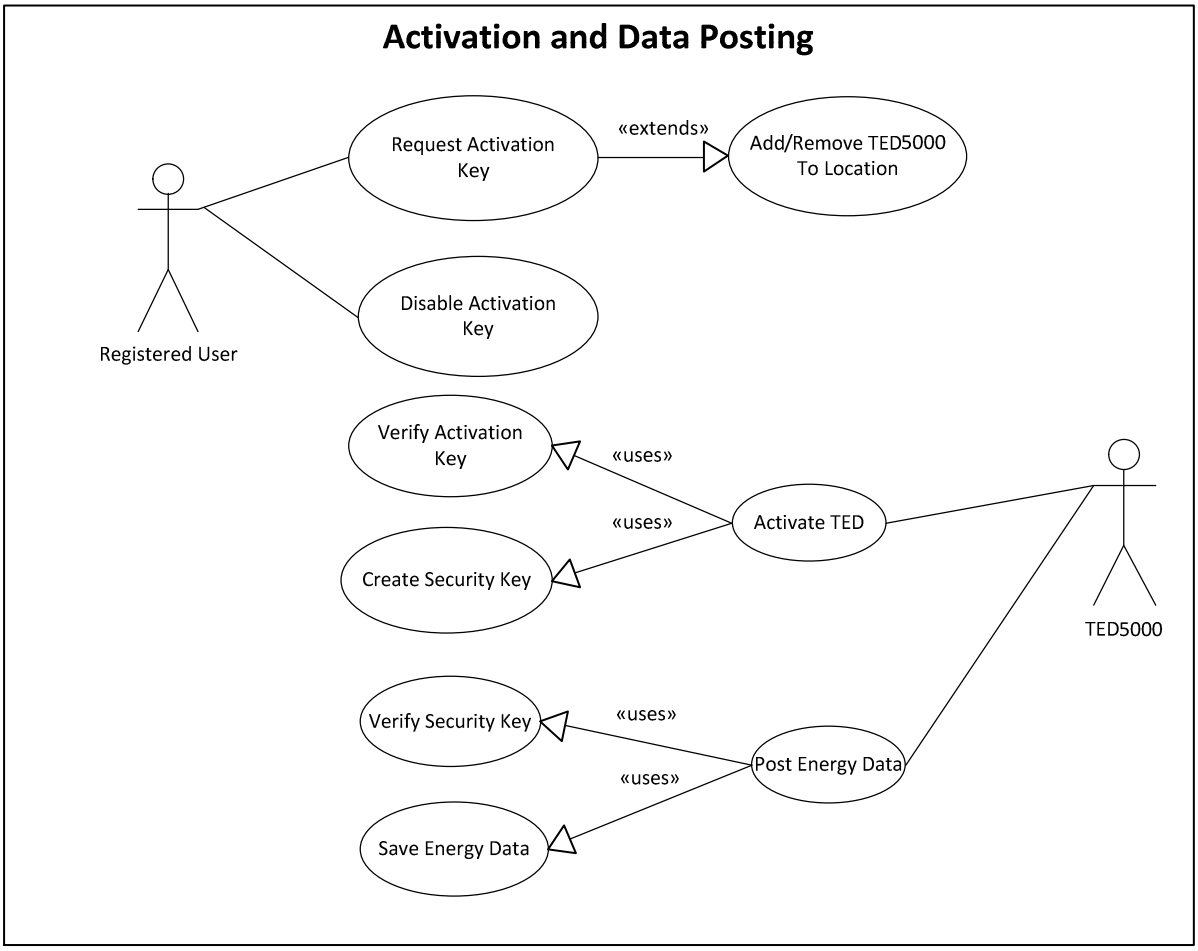
These are the major use cases involved in registering and logging into the system. All users will be able to perform these functions. The Administrator role is just an extension of the registered user role w/ additional privileges in regards to user and group management.

4.1.2. User and Group Management



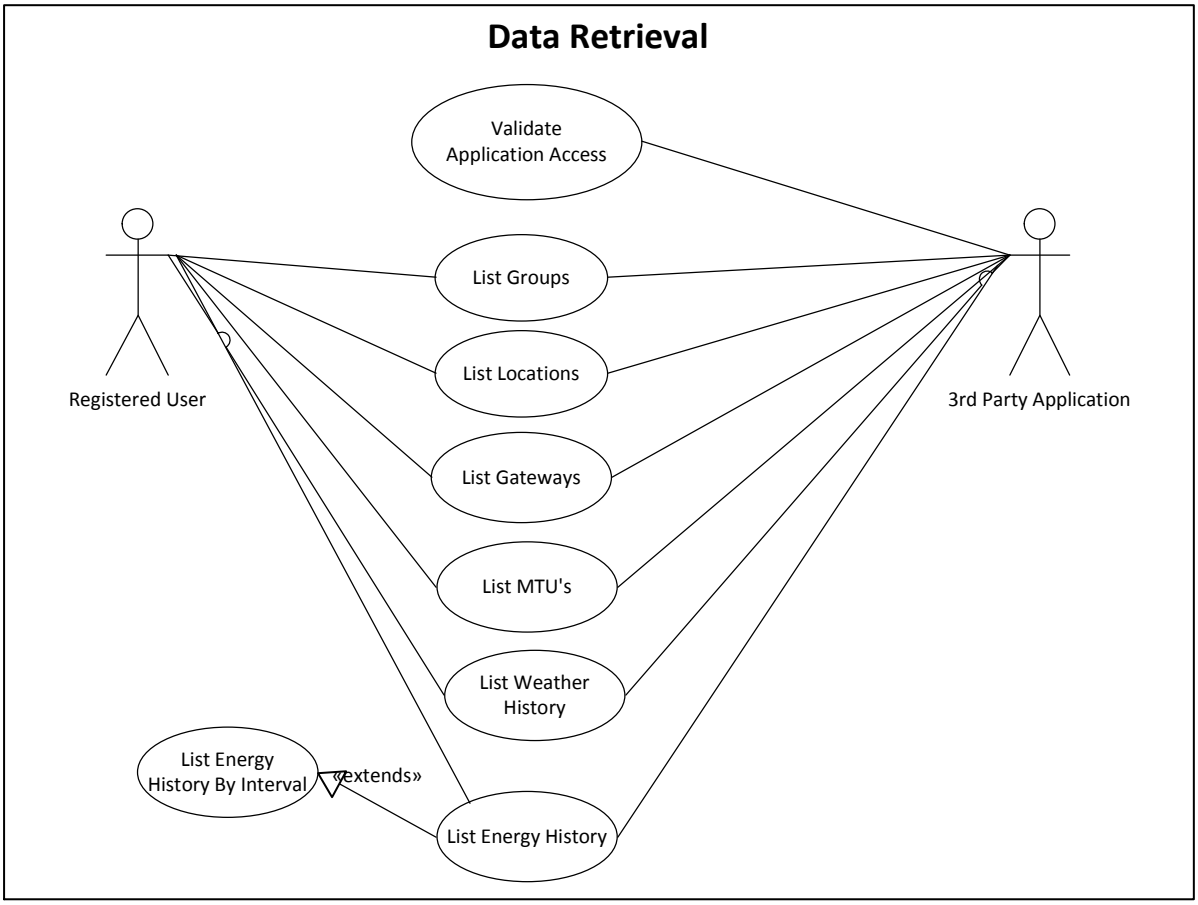
These are the major use cases that allow users to manage their account and to create groups and locations. Please refer to the **Logical View** to see the relationships between TED5000 units, Groups, and Locations.

4.1.3.Activation and Data Posting



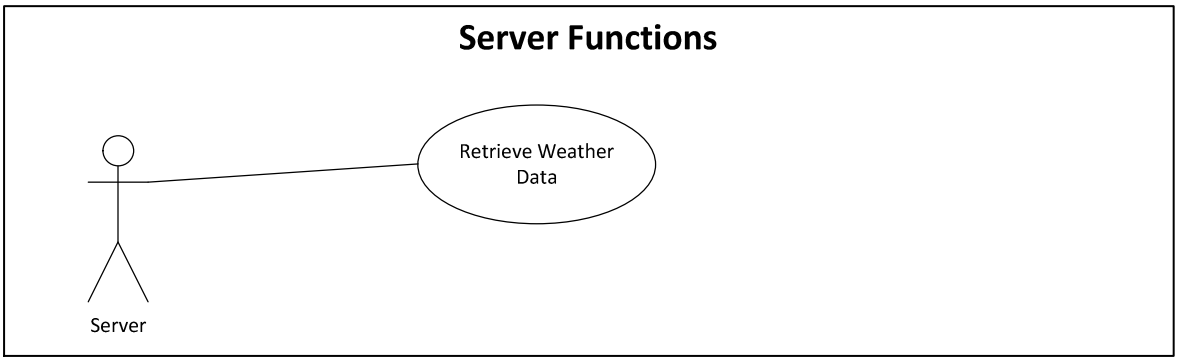
These are the use-cases involved with allowing a TED 5000 to be activated against this server and to post data to it.

4.1.4.Data Retrieval



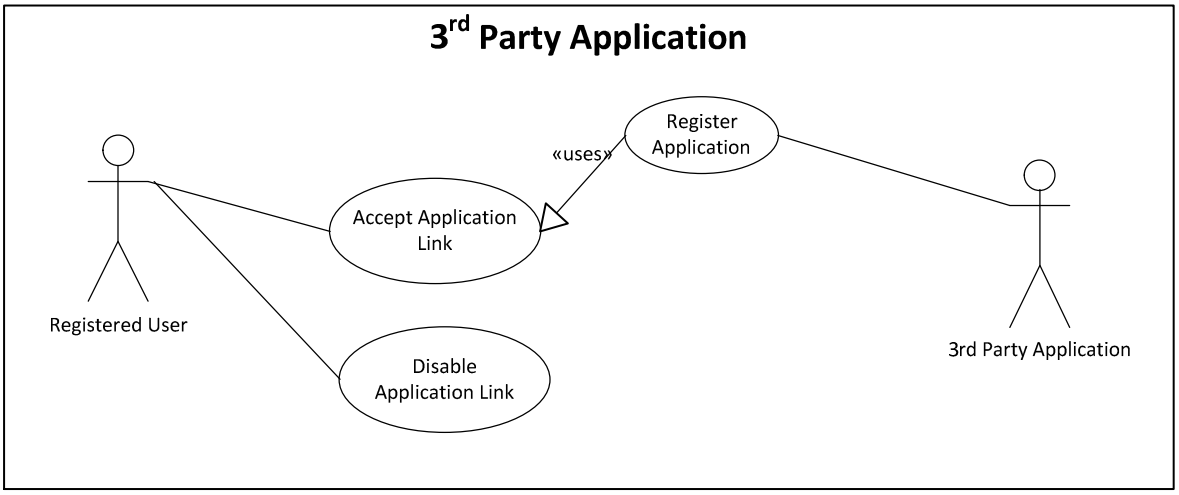
These are the use-cases involved with retrieving energy data from the AggreData system, either through the default GUI or a third-party widget.

4.1.5.Server Functions



Timed server functions to support the Application.

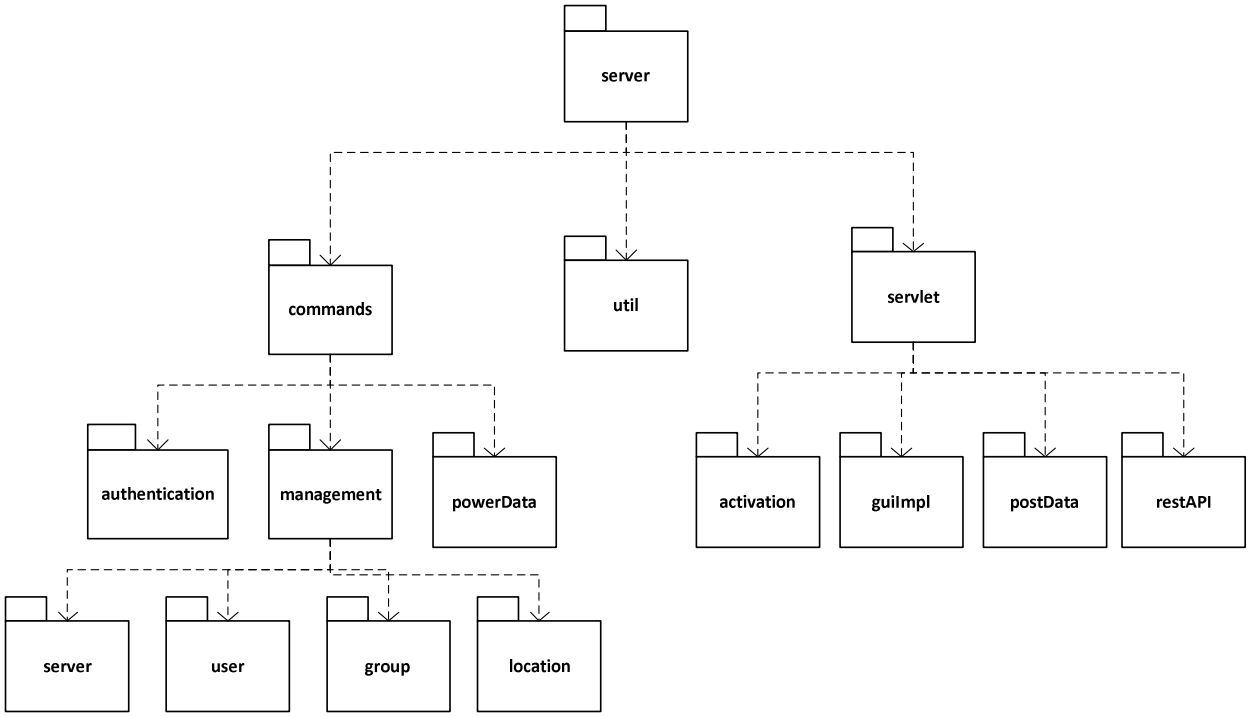
4.1.6. Third Party Application



Use cases involved with allowing a third-party application to access data stored in the AggreData system

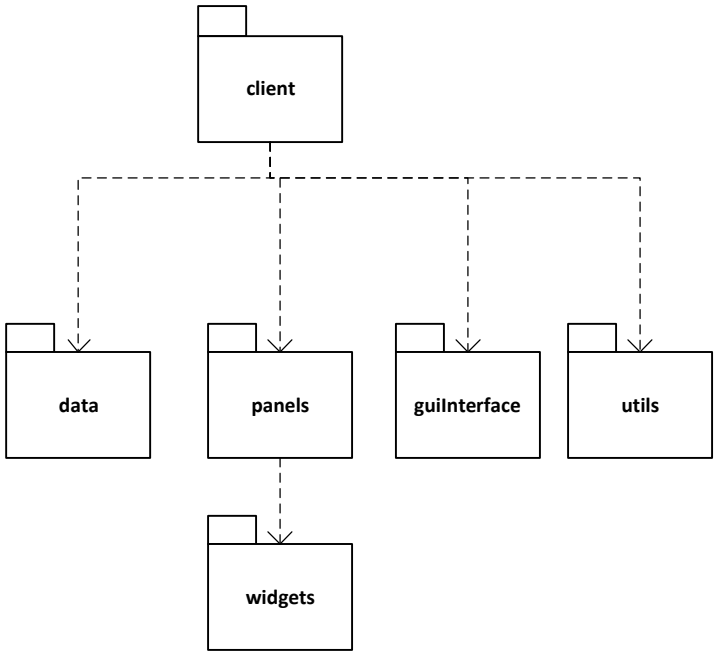
5. Logical View

5.1. Server Packages



- commands – contains the command processor and individual commands that execute the business logic
- commands.authentication – commands related to user/ted5000/3rd party app authentication
- commands.powerData –commands related to posting and retrieving energy data from the database
- management.server – commands related to changing server settings
- management.user – commands related to managing users
- management.group – commands related to managing groups
- management.location – commands related to managing locations
- servlet.activation – servlet code to allow ted5000 apps to enroll with the server
- servlet.guiImpl – GWT interface implementations to allow communication with the client
- servlet.postData – servlet to allow TED5000 to post data to the application
- servlet.restAPI – servlet to allow 3rd party apps to post and retrieve data from the application

5.2. Client Packages

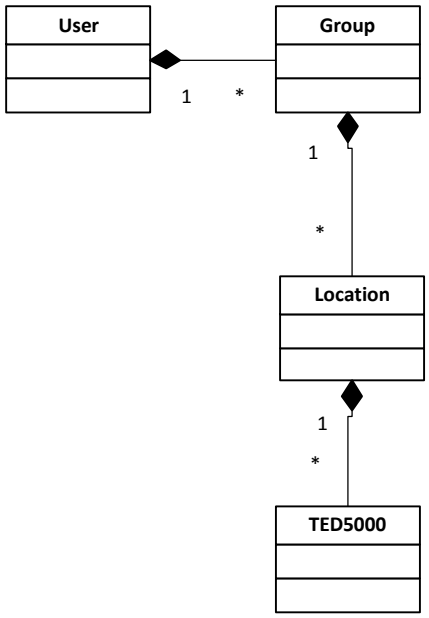


These are the packages that exist on the GWT client side GUI

- data – contains the Data Access Objects (DAO's) used to pass data to/from the server
- panels – the panels that make up the various screens of the client applications
- widgets – reusable widgets that may exist on one or more panels
- guiInterface – Contains the GWT Interfaces required to communicate with the server
- utils – general utility classes

5.3. Entities

The primary objects in the application are Users, Groups, Locations, and TED5000's.

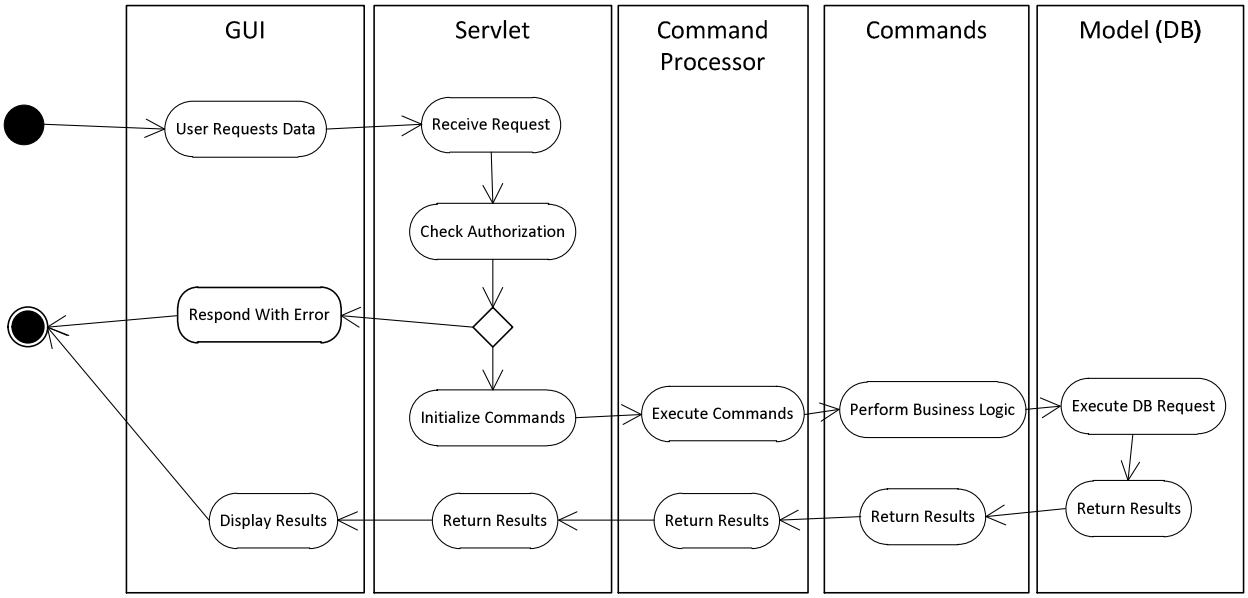


- A user represents an application user
- A user may create one or more groups. Groups contain locations. A location may belong to one or more groups.
- A location contains one or more registered TED5000's. A TED5000 can only belong to one location. All posted historical data is bound to a specific TED5000.

6. Process View

6.1. Application Thread (Activity Flow Example)

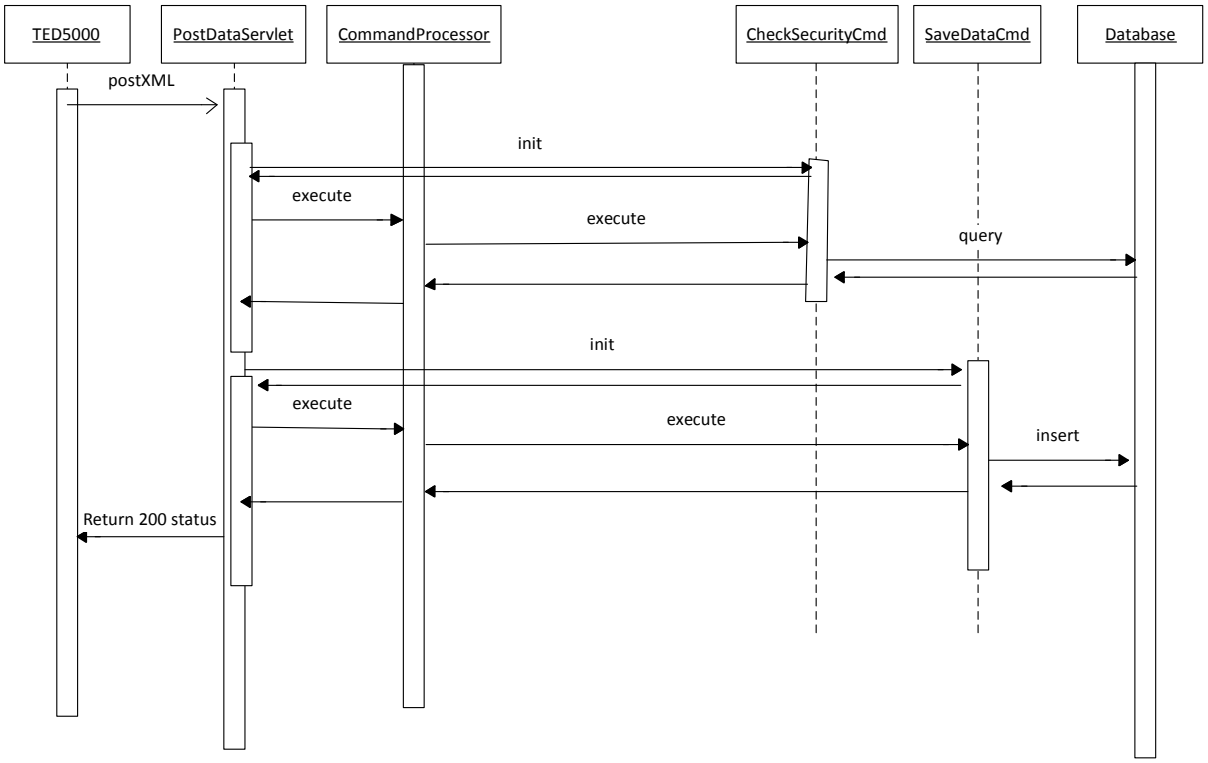
This is a high level overview of how the application thread will generally handle a request.



The GUI will submit a request to a servlet (in this case, a GWT RemoteService will submit a command to its corresponding RemoteServiceServlet. The servlet will then pull the Command Processor out of the session. It will then instantiate one or more commands that will execute the desired business logic. These commands will then be passed to the command processor for execution. Each command will encapsulate a single step of business logic (e.g. create user). The commands themselves will update all model objects in the database should the need arise and pass back any results as Objects via the command processor. The servlet will then pass these results back to the user.

If implemented correctly, the bulk of the business logic code will reside in the commands themselves, and each servlet implementation will only be a few lines of code (minimum amount required to initialize the commands and pass them through the Command Processor.

6.2. Post Data Thread (Command Processor Sequence Diagram Example)



Much like the standard GUI thread, **Post Data** makes use of a servlet that initiates several commands to perform the business logic. In this case:

1. The PostDataServlet parses the XML data
2. The PostDataServlet initializes the CheckSecurityCmd with the Gateway ID and passes it to the CommandProcessor for execution
3. The CommandProcessor calls the execute method on the command
4. The Command then executes a JDBC query to see if the gateway has been registered in the system, if it's still active, and if the supplied security credential is valid. It will return the result as an object indicating the status of the Gateway.
5. Assuming the Gateway is a valid entity in the system, the PostDataServlet then initializes a SaveDataCmd, passing it the energy data and the Gateway ID.
6. The PostDataServlet then passes the SaveDataCmd to the CommandProcessor which in turn calls the execute method on the command.
7. The SaveDataCmd is executed and inserts new rows of data into the database
8. The SaveDataCmd returns the result code to the CommandProcessor which then returns that code to the PostDataServlet
9. The PostDataServlet returns a "200" HTTP response indicating that the data was written successfully.

6.3. List of Process Threads

The threads in the system may invoke one or more servlets to handle the requests. However, each thread is independent of each other, meaning that it would be possible to load balance by thread type if in a clustered environment.

- Main Application Thread – Servlet instance(s) that handle all calls from the GUI portion of the application
- Activation Thread – Servlet instance(s) that handle all requests for TED 5000 activation
- PostData Thread – Servlet instance(s) that handle all requests to post data from TED 5000 units.
- REST API – Servlet instance(s) that handle requests from 3rd party applications
- Weather Thread – Built in application thread (singleton) that requests data from NWS

7. Deployment View

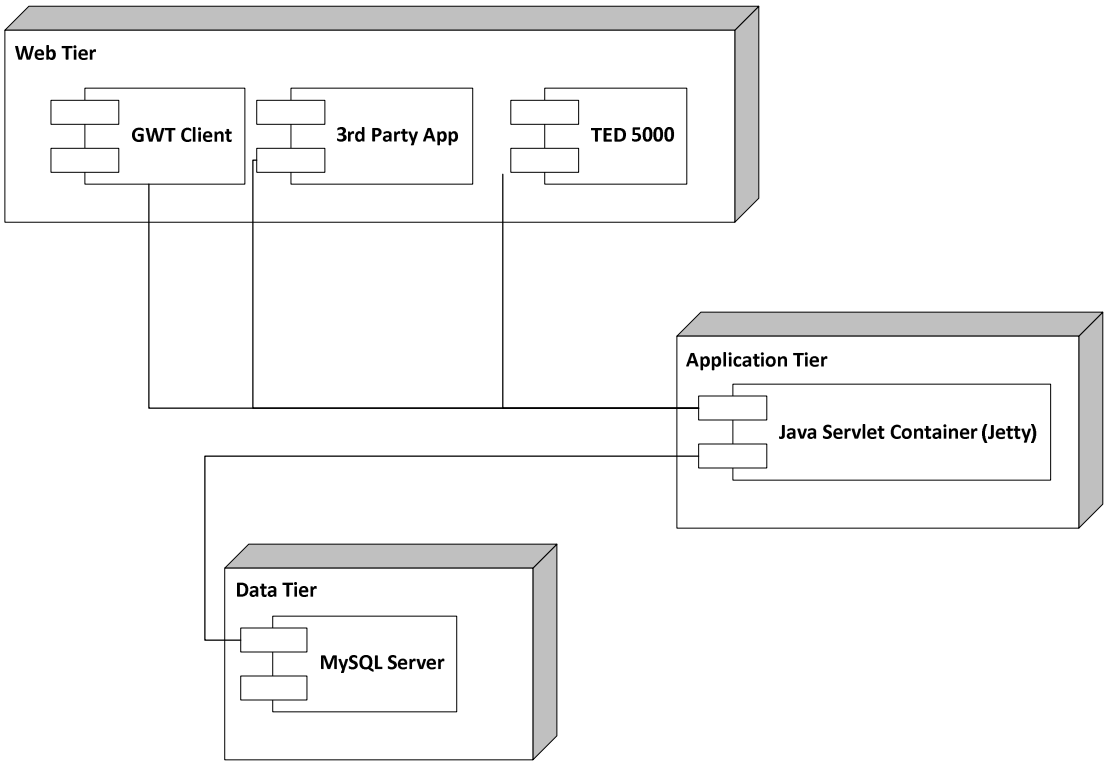
TED AggreData will be built as a WAR web application and an XML configuration file. The WAR file will contain:

- Server-side java classes
- Static web files including images and GWT generated artifacts
- Any dependent libraries required for deployment

The WAR file will be deployed on any standard Java Servlet Container including:

- Apache Tomcat
- Jetty
- JBOSS
- WebLogic

The XML configuration file will need to be visible on the WAR file's CLASSPATH.

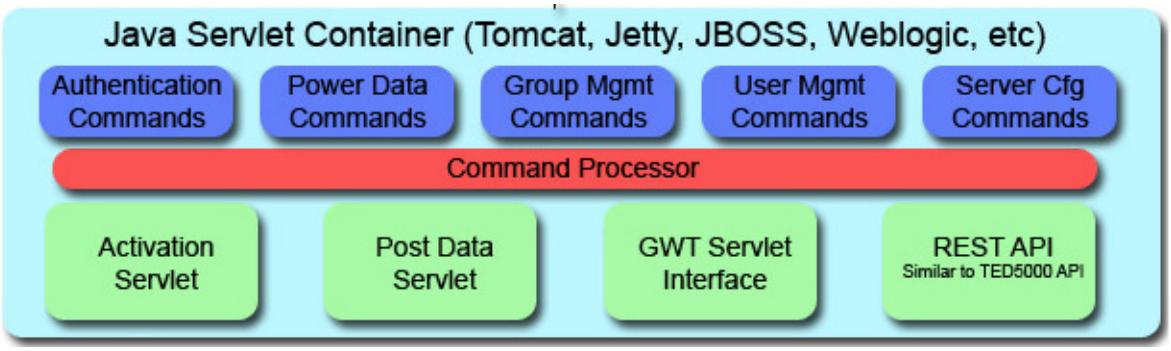


8. Implementation View

8.1. Application Layers

The TED AggreData application will be split into 3 separate layers:

- Client Side UI layer – Implemented as a GWT application
- Servlet Layer /REST API Layer
- Application Logic Layer (commands)

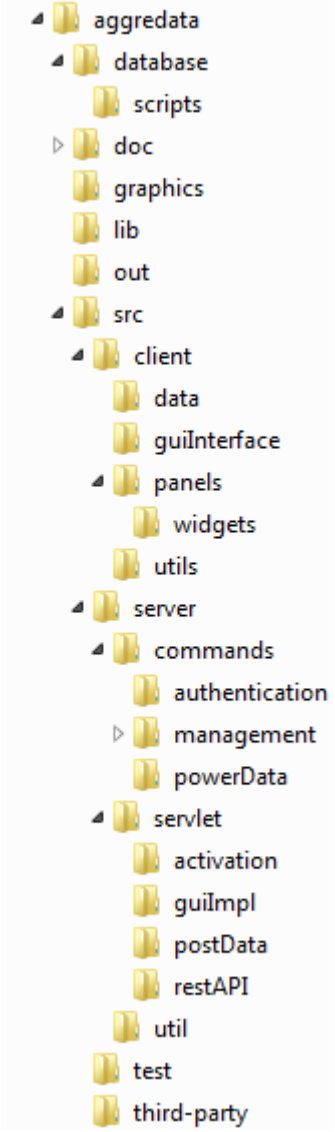


The implementation of the application will follow best practices implementation for standard web applications

- GWT 2.x will be used to implement the User Interface.
- The REST API, Activation and Data Post will be implemented as Servlets
- The *Command Processor Pattern* will be used to implement the controller and model layers. The CommandProcessor itself will serve as the controller, while individual commands will be used to execute logic in the model layer.
- Object persistence will be implemented using direct JDBC connections to connecting via the Bone Connection Pool. Direct JDBC was chosen since very little data will need to be cached and the majority of the workload will require fast access and pass-through to the historical times-tamped energy data.

8.2. File Structure

The project file structure will follow project convention outlined in the article ***Project Conventions for Enterprise Applications Early Access*** *(http://java.sun.com/blueprints/code/projectconventions.html)*



All of the application logic will be stored in the src directory, with the directory structure broken out to the architecturally significant packages of the application.

- **Database** contains the ER diagram and database creation and update scripts for implementing the data model
- **docs** contains supporting architectural documentation (like this SAD).
- **Graphics** contains the raw image files (.psd, .ai) files used to generate the graphics for the application
- **Lib** contains 3rd-party jar files used for this application
- **Out** contains the generated WAR artifacts
- **Src/client** contains all client-side source code including GWT components, DAO objects for communication pass-through, and generic client-side utility classes
- **Src/server** contains all server-side code including business logic commands, servlet, and related utilities
- **Test** contains the suite of JUNIT tests used for unit and functional testing of the application
- **Third-party** contains source code for sample 3rd party widgets compatible with this application
- **Web** contains static files and war-deployment files for the application

8.3. Database Model E-R Diagram