CS307 -- Software Engineering

Design Document

Team 17

Aaron Peters

Ben Whorley

Charles Cho

Paul Lohmuller

Zhihao Hu

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **Title** | **Page** |
| PURPOSE | 3 |
| DESIGN OUTLINE | 4 |
| DESIGN ISSUES | 6 |
| DESIGN DETAILS | 8 |

1. Purpose

Currently, seismic events occur all over the United States every day. Seismic sensors are developed to capture these seismic events. Teams that are dedicated to developing these seismic sensors would like to know where to test them to know that they are accurately picking up seismic activity. Currently, there is no easy-to-use map that shows areas of little seismic activity where it would be ideal to test these sensors. Our team will implement a system to fix this.

This design document thoroughly describes our functional requirements.

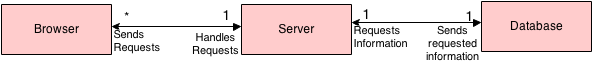
1. Data is interpreted from public seismic activity monitors
   1. It will be displayed as a graphical data plot.
   2. It will be stored into a database.
2. Users can look up previous events using a database
   1. Users will search by date.
   2. Users will search by magnitude.
   3. Users will search by sensors.
   4. Users will search by locations.
3. The front-end interface is a map application
   1. On the map, users are able to see the epicenter of an event.
   2. On the map, users are able to see the radius of an event.
   3. On the map, users are able to see which areas on the map are least active seismically and which areas are most active seismically so they can determine the best areas to test new sensors.
   4. On the map, users are able to select and deselect certain seismic sensors so they can view certain regions of seismic activity.
4. The graphical interface will be user-friendly
   1. It will be available online.
   2. Navigation will be by mouse.
   3. Users will manipulate menus, buttons, and toggle switches.
5. Design Outline

Our project will mainly be a browser application that allows users to see areas of different seismic activity by viewing a map which is color-coded. Our system will use the Client-Server model where our server will be connected to many different clients each viewing this seismic activity map. Our server will implement the MVC organization method. It will need to be connected to a database so that it can service all requested information, and have somewhere to store the large amount of previous seismic data.

1. Browser
   1. The browser will be where the user’s main interaction with our system will be.
   2. The browser will display a map where the user sees the general seismic data.
   3. The user will be shown UI features where they can request different types of information, such as previous seismic events or the location of seismic sensors.
2. Server
   1. The server will be the middle-man between the browser and the database; it will handle all traffic.
   2. The server will generate requested web pages to display in the browser View.
   3. The server will request data from the database if needed for the webpage it is trying to generate.
   4. Any back-end calculations or algorithms will be run on the server.
3. Database
   1. The database will store all major data we will need for this project.
   2. Typical data will be previous seismic events and seismic sensors.

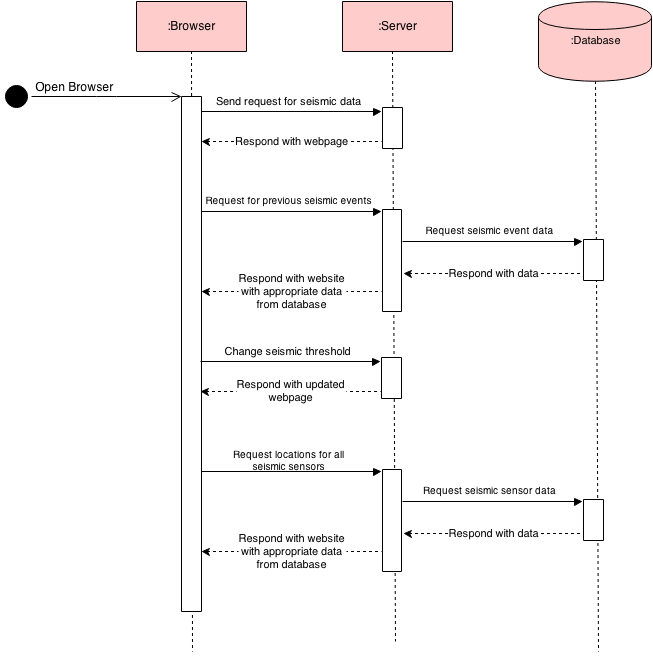
**Figure 2.1 - High level overview of our system**

We will have many browsers that are connected to a central server. Each browser will send a request when the user needs new information (i.e. previous seismic events, seismic stations). The server will then decide what action it needs to perform. It may simply need to upload the web page, or it may need to send a request to the database for missing information.



**Figure 2.2 - Broad overview of typical sequence of events**

This figure shows a typical sequence of events that may occur when a user accesses our browser. The general pattern is that the browser requests information from the server. If the server has the means to generate the webpage, it will do so and return that page. If not, it will access the database for the needed data and then return the finished webpage.



1. Design Issues

Functional Issues:

**Issue 1**: Browser Implementation Selection

Option 1: HTML

Option 2: CSS

Option 3: JavaScript

Option 4: Vaadin

Decision:

We chose Option 4 for two reasons:(1) One of our team member had experience of developing a web page with this java implementation. (2) It is easy to use Java, because all the team members are very experienced with Java coding.

**Issue 2**: Threshold for seismic events

Option 1: User defined

Option 2: Developer set default

Decision:

We decided that having both options is viable since (1) depending on the user’s level of technical understanding, they may or may not want to set the threshold themselves, so there will be a default threshold for inexperienced users, and (2) an option to change if the user so desires.

**Issue 3**: Map API Decision

Option 1: Google

Option 2: mapQuest

Option 3: Yahoo

Option 4: Bing

Option 5: OpenMap

Decision:

Google maps is the most widely used map API now. Since Google maps is very advanced and it can be implemented using Java easily. So after discussion we decide to use Google map as the API Decision.

**Issue 4**: Data search function

Option 1: Search seismic events data by date

Option 2: Search seismic events data by magnitude of events

Option 3: Search seismic events data by particular sensor

Option 4: Search seismic events data by location

Option 5: Search seismic events data by administrator (the sensors he or she owns)

Option 6: Search seismic events data by searching history

Decision:

We decided to use Options 1 ,2 and 4. Normal users don’t need to search the data by administrators, and also implementing Option 5 required much more work including user data, user activity storage. So Option 5 will be abandoned. And compare to Option 1 to Option 4, Option 5 and 6 is much less important.

Non-Functional Issues:

**Issue 5**: Database Architecture

Option 1: mySQL

Option 2: SQLlite

Option 3: Microsoft Database Management

Decision:

We will use mySQL as our database architecture, because it is easy to understand and can be well developed with Vaadin.

**Issue 6**: Client Browser Display Options

Option 1: Tabs

Option 2: Buttons

Option 3: Menu Options

Option 4: Context Popups (right click)

Decision:

We decided that all of these options are viable, and would implement them if possible, but menu options would be the simplest means to implement.

**Issue 7**: Administrator or Normal User

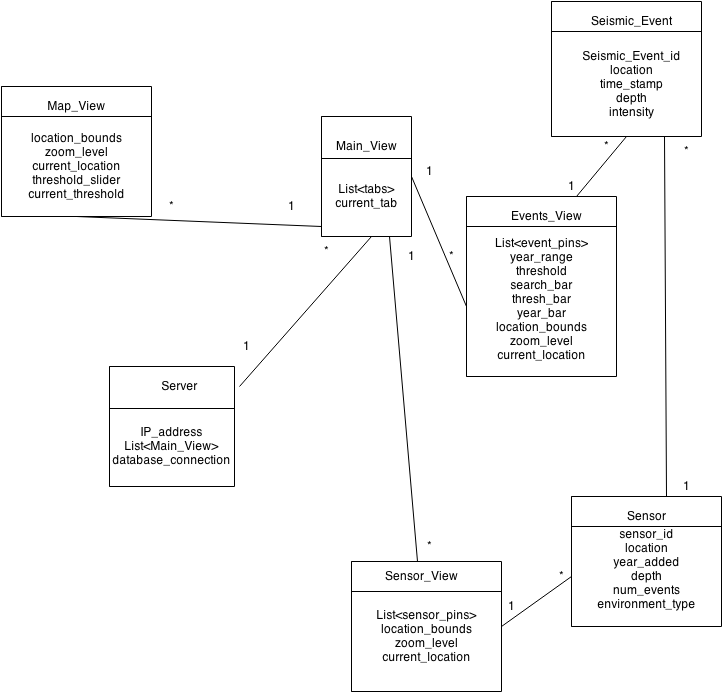
Option 1: Adding sensors will be restricted to the administrator

Option 2: All users will be able to add new sensors

Decision:

We decided for Option 1, only administrators can add new sensors. Since if all the users can add sensors, data will not be very reliable from the new adding sensors. And also for security reasons, it’s better for only administrator to add sensors.

1. Design Details
2. Class Design



1. Description of classes and models

Server:

● Basic class for web that will store information such as IP address, database connection and list of Main\_View Objects.

● Server can connect to database. So any class that needs to get information from the database will call the Server class and access the information it needs.

● Server will call Main\_View when any user accesses the data

Main\_View:

● Main\_View is the basic UI class that connect to the three main tabs of different pages(map, sensor, events).

● Access three tab classes: Map\_View, Sensor\_View, Event\_View. And display certain content from three classes

● Any change of tabs will lead this class to access different View classes

Map\_View:

● One of the View classes will display the map and show different colored areas based on the frequency and intensity of seismic events

● Will be called by Main\_View (default tab) when user opens the page for the first time.

● This can fill the colored area using a slider of thresholds, and it will change the colored area currently showing on the map.

● Variable zoom level will control the current scope of colored areas.

Sensor:

● The Sensor class represents each seismic activity monitor and is used

by Sensor\_View to mark the geographic location of each sensor.

● References Seismic\_Event data obtained by each sensor.

● Holds information concerning the sensor such as location, local

environment, number of events, and year added.

Sensor\_View:

● Initially setup with full Map\_View features

● Display the pins of sensors on the map, also can access information of sensor by clicking the pin of that sensor

● For each pin’s information, this class will retrieve the informations from the Sensor class

Seismic\_Event:

Contains all the relevant information pertaining to seismic events collected by the sensors.

● Stores the location, timestamp, depth and intensity of each earthquake in the database

● Provides information to Event\_View when necessary.

Events\_View:

● Initially setup with full Map\_View features

● This view class displays pins for each Seismic\_Event on the map at the

locations of respective Sensor objects.

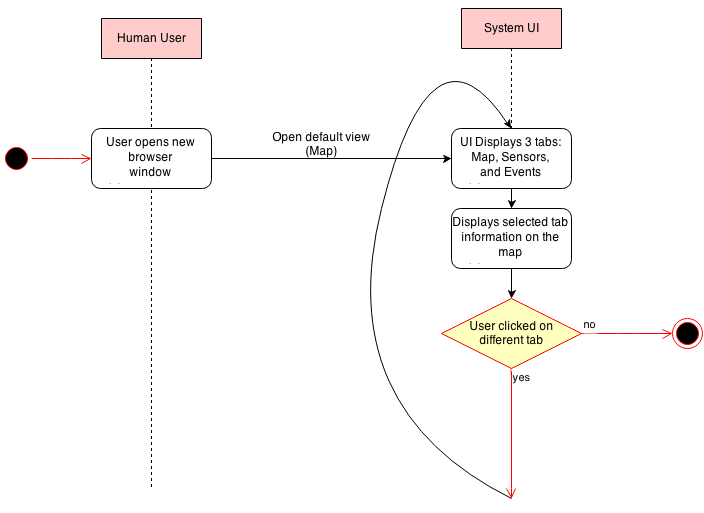
● Offers sort and search options based on intensity, time of event, and

location.

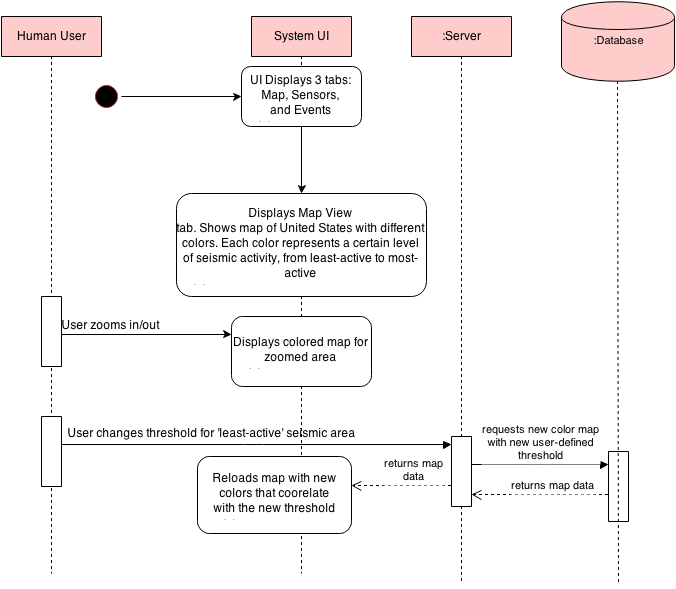
● Show more detailed information of seismic events when clicking the pin of any seismic event.

1. Sequence Diagrams

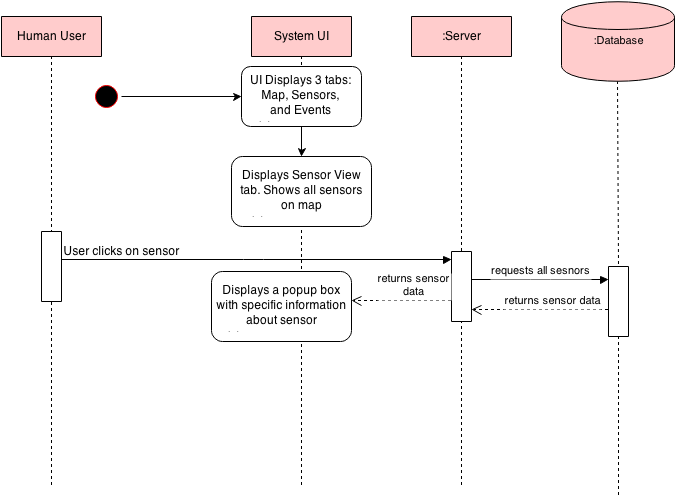
**Figure 4.1** - Sequence of events when user first opens browser. The tab they will be defaulted to is the Map View



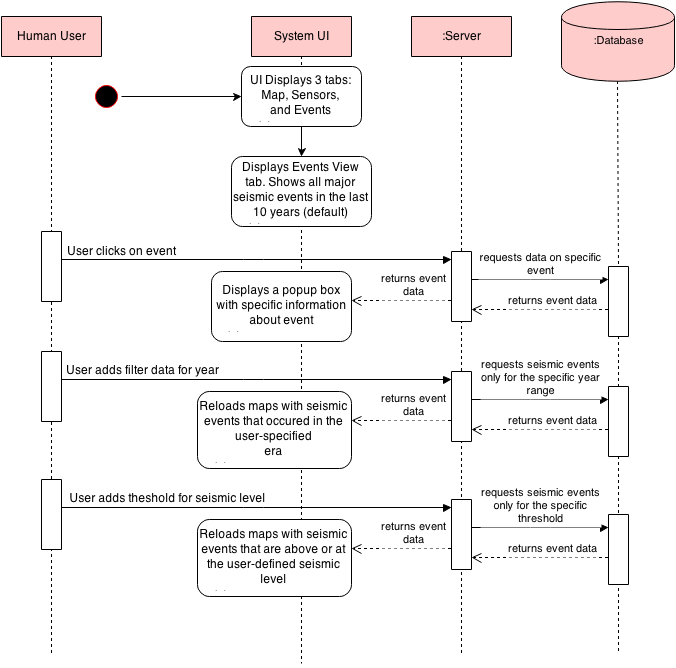
**Figure 4.2** - Sequence of events when a user clicks on Map View tab. The user can zoom in and out, or change the threshold for the colored areas.



**Figure 4.3** - Sequence of events when a user clicks on Sensor View tab. The user can click on a specific sensor which could call the server and database for that sensor’s information.



**Figure 4.4** - Sequence of events when a user clicks on Events View tab. The user can click on a specific event which would display information about that event. The user can also filter events by attributes such as location, intensity, time, and sensor.



1. Mock-Up UI for Event View

