NEIGHBORHOOD INFORMANT

A Professional Report about a resource for Chicagoland homes data in a user-friendly application.

Prepared by:
Aiwan Hazari
Jay Patel
Dhrumil Patel
Deven Patel

Created in CS 442 in May 2017 at the University of Illinois at Chicago

Table of Contents

I. Project Description

A. Project Overview

Neighborhood Informant is a desktop application that provides ease to Chicagoland residents by providing a single application of various Chicagoland data. This application will gather real and accurate data straight from the City of Chicago data portal website. Getting data about an area has become a huge hassle this days; there's a need to visit different websites to get different data, and most of the time it’s out of date. Furthermore, the City of Chicago (website) does not provide a good graphical user interface for users. This application will have an easy-to-use user interface that will allow a user to pick a location and neighborhood and receive relevant feedback about it, such as crime, landlords, etc. Neighborhood Informant will be a one-stop for a multitude of accurate Chicagoland information.

The purpose of the project is to give users a tool for easily identifying different aspects of the neighborhood or a particular area. Currently, it only gives information about the crime rates and problematic landlords, but other information such as per capita income can be added in the future. The goal of the application is let users get all the information in the application without going to other websites or tools to get information.

No technical experience is required to use this application. It has a very simple graphical user interface. It can be used by everyone from anywhere. No location restrictions are present. The application is made in Java for universal compatibility. Making it in C# or HTML would have broken compatibility with some systems. Furthermore, making a

website was not a part of the initial goals of the project, and therefore would not have adhered to the requirements. The application can be used on Windows, Mac, and Linux as well.

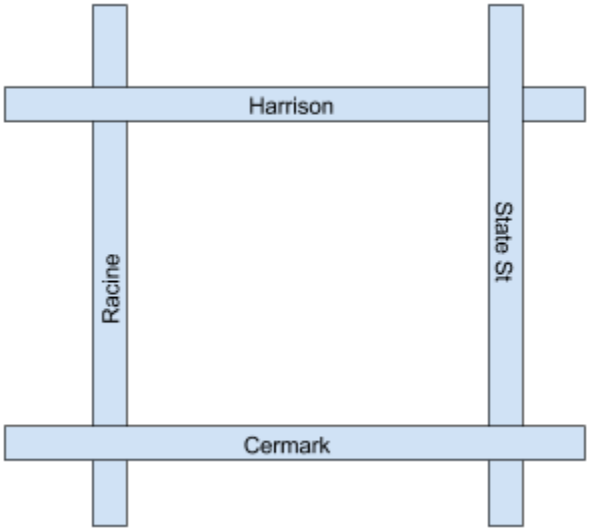
B. Users and Features

This application can be used by everyone, not just for people buying houses. Users can also track the crime points near their house. The application provides real-time data so they know which area is safe and which is not. On top of that, users can provide their information as well. If a user was the victim of a crime, he/she can enter the address, and the location (Lat, Long, and Addr) and the crime will get saved in the database. Then this data can be visible to everyone in real-time.

Neighborhood Informant gets information directly from the City of Chicago to get precise location-based data. It regularly fetches data from their database. And then the crime points show up immediately on the GUI. We also added the data to our Firebase database so new points can be visible, that is, the crimes that are not reported in the City of Chicago database.

Furthermore, this application only enables data within a small radius of Chicago. The figure below shows the intersection streets this application shows (Harrison, State, Racine, and

Cermark). This allows the application to have full functionality of all features, without sacrificing speed to factor in gigabytes of memory.



II. Collaboration Tools

Asana was the main collaboration tool for project maintenance. Asana has many features such as creating stories, assigning tasks, calendar view, sprint information, instant messaging, group chatting, and file upload. We all would know what was the progress was for the application, and whether a task was finished or not. If the task wasn't finished, we could "claim" it. Otherwise we could create a new task to work on. We also sent out group messages that would immediately email everyone in the group. Here we discussed tasks to include and information about the sprints. This was useful for interacting between everyone.

WhatsApp was another collaboration tool we used. WhatsApp is an instant messaging application, which was great for immediate interactions between team members. Here we would discussed times we would meet up, changes any group member made to the application, and assignment/tasks distribution. Since everyone used WhatsApp regularly, this was a great tool to fit into our development process.

Git was another collaboration tool we used. Git allowed us to upload our code and data files directly onto a repository. Then, as team members changed their code, they could upload it and everyone could access the latest code (without overwriting their own changes). We used our own Git repository for the main code (https://github.com/Dhrumil95/CS442), as well as the class assigned repository for assignments and progress reports.

III. Requirements

In order to setup this application, the user needs to download all the application files we created this semester and install a few necessities on their machine. The machine needs to have at least 2GB of memory in order to successfully download every aspect of the program. It also need to have the Java Environment installed, preferably with the newest version available. The machine's Wi-Fi must also enabled, in order to access the Google Maps API and Firebase database.

The user will also need access to the JXBrowser library. JXBrowser allows integration from a desktop application to a web server in order to access websites with an easy-to-use browser. The library can be found directly on the JXBrowser website:

https://www.teamdev.com/jxbrowser. The user will need to download the JXBrowser software and register for an account.

Once these steps are completed, the user must attain a license for the JXBrowser library. This can be done via 3 different ways. One, the user can attain a free-trial license for 30 days. Second, the user can register with a student or business account and use the license (for free) for the duration of their use. Third, the user can just purchase the license. Once the user has access to the JXBrowser library license, the license.jar file should be downlaoded and placed inside of the "jxbrowser ../lib/" folder.

In order to run this application, the user can use some kind of IDE for instant access to the application. The IDE must be a Java-enabled IDE, such as IntelliJ or Eclipse. If there is no IDE installed, the application can also be run through a terminal window (with the available Jar file, ProjectBrowser.jar).
The user can also compile the project via a terminal window and attain a Jar file by typing the following:

1. javac ProjectBrowser.java
2. java ProjectBrowser

IV. Design

Every class is connected to each other. A new instance is created for each view. We also used event-driven programming for this application. This allows the user to determine what the application will perform. Every panel has a instance dedicated to it, so when a user clicks on a button, the desired instance is created and the panel becomes visible onto the

frame. All the files (described and not described) are available in the GitHub repository: https://github.com/Dhrumil95/CS442.

Below is a description of the files/classes:

Class	Description
CityWeb	Creates a new ProjectBrowser instance with the required attributes *windowTitle = "City of Chicago Official", fileName = "https://data.cityofchicago.org"*
CrimeMap	Creates a new ProjectBrowser instance with another view(panel). This creates a new Utilities instance and calls the getFilePath method to get the map file. The file is named crimemap.html. This uses the attributes *windowTitle = "City of Chicago Map, fileName = this.fileName*
Data	Gets data from the Firebase database. Gets year, longitude, latitude, address, and ID.
Database	Connects the application to the Firebase database. Makes Firebase calls with a dedicated link to database (https://cs442-61428.Firebaseio.com). This class also stores new information through *addNewData()* method.
JSON	Creates a new JSON file for the data retrieved from the Firebase database. There are also sanity checking methods involved in this class. Those are: isInteger(), isDouble(), _isLatLonValid(), isLatitudeValid(), isLongtitudeValid(). They all have parameters according to their functions.
LandLord	Creates a new ProjectBrowser instance with another view(panel), for for panel viewing. This uses the map file to show the landlord data. This data consists of landlords who have been reported as problematic.
ProjectBrowser	The main Graphical User Interface. Includes buttons for all features, crime data, landlord data, etc. Also includes buttons such as add back button and panel hiding.
Mainpage	Shows the points on a map and refreshes at a very short interval to create real time viewing.
Utilities	Used to get file paths, mainly file paths of the data files.

We inherited the classes from a superclass to make the code less complex, readable, and understandable. Avoiding duplication of code was our top priority, therefore writing reusable code was quite important. We started off with the inheritance of different classes. Then as we refactored our code, we applied design patterns.

The Singleton Design Pattern was the pattern we mainly used. This pattern was helpful for our event-driven programming approach. It made sure only one instance of the main GUI was created and therefore only one instance of access to the databases. Without the Singleton Pattern our code would have become very slow if the user was given the opportunity to create multiple instances.

V. Programming Practices

We used Extreme Programming throughout the development of this application. Our process of Extreme Programming consisted of 4 different sprints and one test sprint. Within each sprint, we used one week to program new code, and one week to test our new and old code as well as integrate the two together.

We used the Test Driven Development (TDD) method because the functionality of Neighborhood Informant was quite straightforward. It also fit perfectly well into our Extreme Programming scrum practices. We were able to divide up the functions and events we were supposed to work on within each sprint with ease. Therefore, each of us created a series of test unit throughout production, and tested the applicable functions. Our process followed the TDD guidelines, as shown in the image below.

We also used Pair Programming throughout development. Following pair programming guidelines significantly lowered our learning curve for new development tools. None of us had had any experience using Firebase, Asana, or JXBrowser; therefore pair programming made it a lot easier to learn and integrate these tools into a fully functional program. After mid-production though, pair programming no longer seemed necessary, since we all had assigned tasks that were not too difficult to do. At that point, we switched over to working on our own tasks independently and collaborating together via Git and Asana.

We have used many design patterns in creating the project. Factory method was used to create multiple instance of the JXBrowser, since they all shared similar characteristics. A builder pattern was used to create the JXBrowser (some instance only required the web URL, such as the City of Chicago website). We have the MVC pattern as well, as soon the data gets changed on the database, the changes immediately reflect back on the screen.

VI. Testing

Because we used Extreme Programming, testing was a common practice for us. We added code for one half of a sprint and then tested the next half. The way we tested our code mostly adhered to Test Driven Development guidelines, as shown in the figure below. We wrote our new code. Then wrote tests for our code. As long as these tests failed, we continuously kept fixing our code to make the tests pass. Then when a test would pass, we would refactor this code and clean it up to improve its efficiency.

Our plans to create Unit tests for the functions and events started in the early stages and continued onto the last day. To avoid refactoring of the code at the end, running JUnit tests after implementing new functionalities was the most efficient method for us. Because of these practices, we went from inheritance to design patterns to more efficient methods of data manipulation. Further, not a lot of refactoring was required at final stages of development.

VII. Project Issues

Even though we used GitHub for collaboration and code refactoring, getting it together was quite difficult. At the beginning we overwrote some commits, pulled wrong commits, pushed without pulling the latest code. But as the development progressed, we all got familiar with the GitHub collaboration. When a new commit was pushed, we would send

messages in our WhatsApp group chat to make sure everyone knew that a new commit was pushed. This would let everyone pull the latest code with bug fixes and refactoring.

At the beginning we also had unrealistic goals to some extent, missed some tasks in first sprints, and had difficulty making the graphical user interface user friendly and simple. We also juggled around the tools to use, and which tools would be perfect for this project. We finally decided to use the developing tools describe earlier in the document.

Data was a major issue in the beginning. The City of Chicago database only allowed *all* the data to be downloaded at once. This mean that we had tens of gigabytes of data files to loop through, which caused our program to be extremely slow. Because of the slow speed, a lot of functionalities did not work. In order to manage the data and functionalities, we had to instead use fake, and a small amount of, data. Eventually, with Tanima's suggestion, we decided to use just a small amount of Chicago data for our entire development and production. This made it so that the application was quick and implemented all of our functionalities

Implementing the Google Maps API was also a challenge. Initially we spent days trying to implement Google Maps with Java. In time we realized that it was not equipped to handle Java. Therefore we used the given HTML/CSS/Javascript approach from the Google Maps documentation. This was simple to implement, however it added the challenge of integrating this into our desktop application.

After coding HTML implementation we contemplated making Neighborhood Informant into a web application. This did not adhere to the original requirements of the project though. Therefore, we did research on other tools to use to integrate HTML and Java on a desktop application. In time we landed on JXBrowser, which solved all our compatibility issues.

Getting familiar with JXBrowser and Java JFrame integration was also difficult. Firebase calls sometimes worked and sometimes didn't with these frameworks. In time and as we debugged, researched, and read more documentation, we came up with the fixes to these issues as we got more familiar with the tools.

User is greeted with the following screen:

User has 5 options to choose from. Each of them has different functionality.

-- Crime Rates

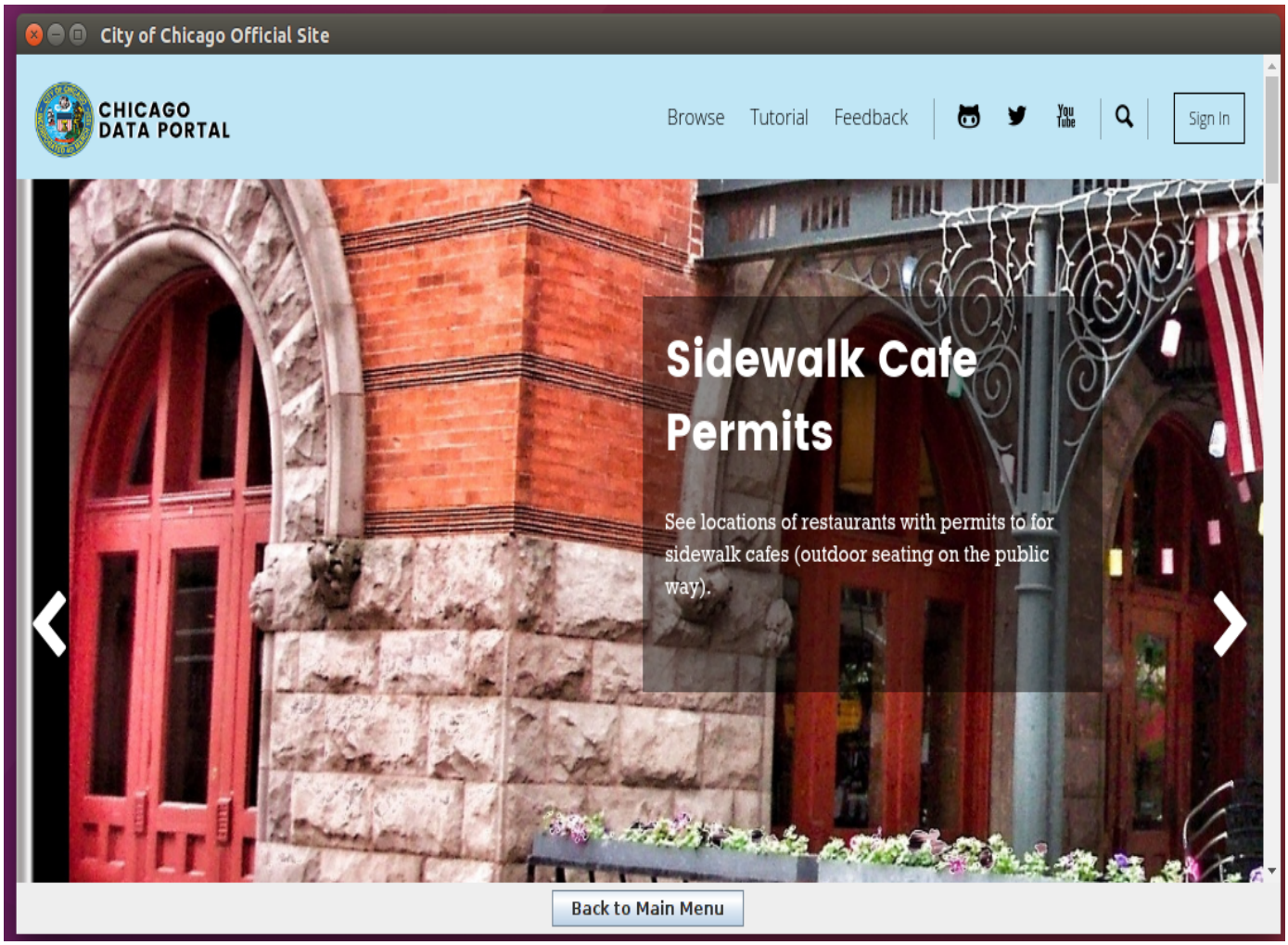
-- Avg Income/Capita

-- Problematic Landlords

-- City of Chicago Homepage

Below are the screenshots when each button is clicked.

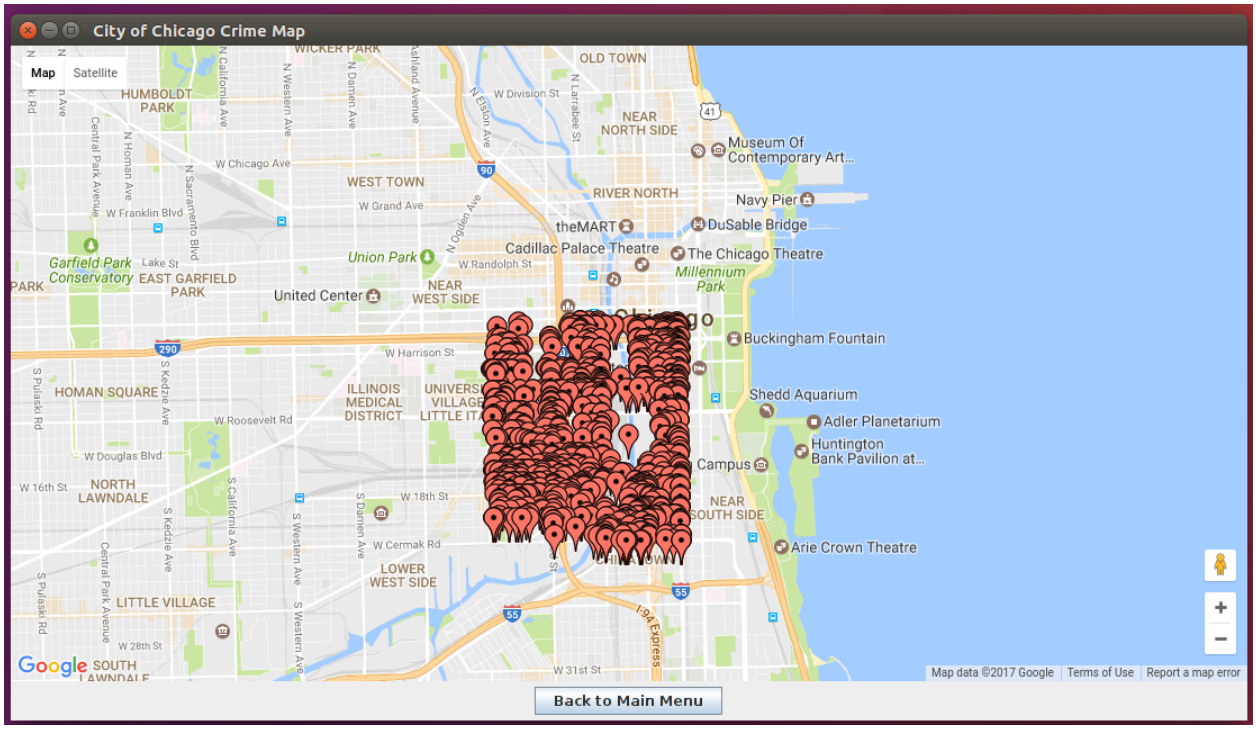
Clicking on City of Chicago Homepage takes the user the official data website.



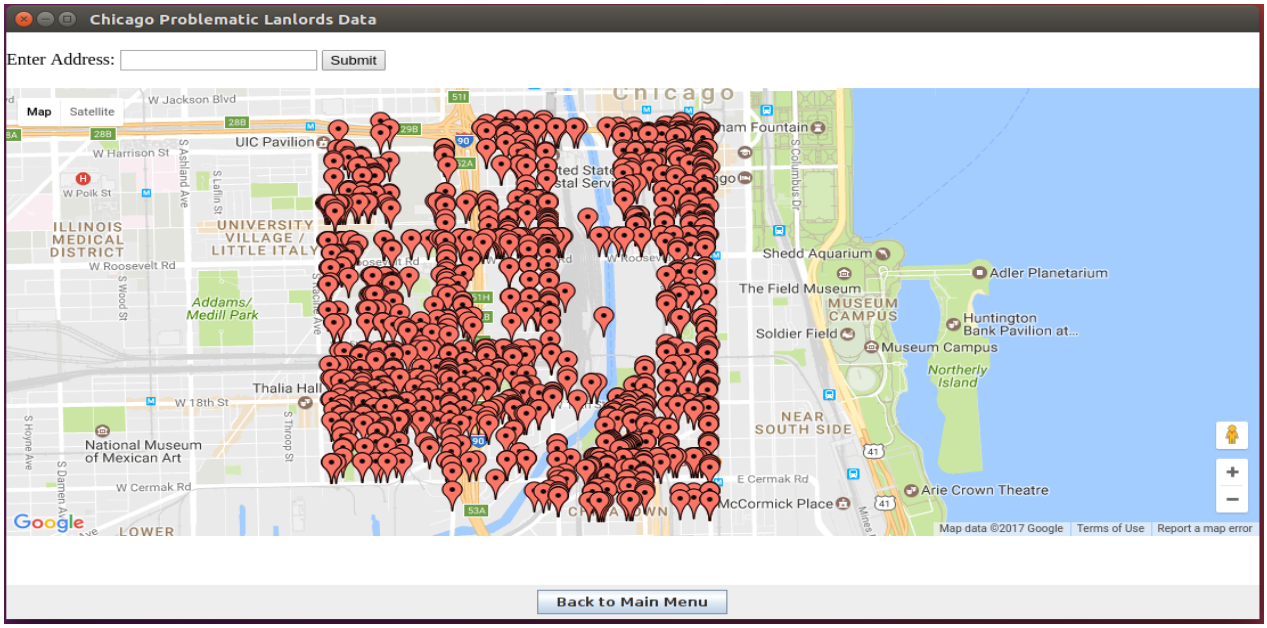
Every panel has ‘Back to Main Menu’ button to go back.

You can browse different data from here. It also provides the user with tutorial and other support area. You can sign in as well to save your information.

Clicking on City of Chicago Crime Map shows the following screen. You can zoom in and zoom out as per your requirements



When clicked on “Chicago Problematic Landlords Data”, user is greeted with the following screen:



Notice, there is a Text with the label ‘Enter Address’. User can enter the address there. The location will get saved in the map in the real time as shown in the pic below.
It will be updated to every application in real time.