COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

GIStice League

Clare Garski
Jordan Ehrlich
Wesley Golliher

Revised December 4, 2022

CSCI 370 Fall 2022

Prof. Painter-Wakefield

Table 1: Revision history

Revision	Date	Comments
New	8/30/2022	Completed Sections: I. Introduction II. Functional Requirements III. Non-Functional Requirements IV. Risks V. Definition of Done
Rev 2	9/18/2022	Completed Sections: VI. System Architecture Updated Sections: I. Introduction
Rev 3	10/21/2022	Completed Sections: VII. Software Test and Quality VIII. Project Ethical Considerations
Rev 4	11/10/2022	Completed Sections: IX. Results X. Future Work XI. Lessons Learned XII. Team Profile
Rev 5	11/29/2022	Completed Sections: VII. Technical Design References Appendix A – Key Terms Updated Sections: I. Introduction II. Functional Requirements III. Non-Functional Requirements IV. Risks V. Definition of Done VI. System Architecture VIII. Software Test and Quality IX. Project Ethical Considerations X. Results XI. Future Work XII. Lessons Learned
Final Rev	12/4/2022	Updated Sections: I. Introduction II. Functional Requirements III. Non-Functional Requirements IV. Risks V. Definition of Done VI. System Architecture VII. Technical Design VIII. Software Test and Quality IX. Project Ethical Considerations X. Results XI. Future Work XII. Lessons Learned Appendix A – Key Terms

Table of Contents

I. Introduction

The goal for our project is to create an interactive maps software that allows visitors to the Colorado School of Mines library and maps room to place a digital pin on a map of Colorado, the United States, and the world based on where they live. The software collects the location information and displays maps relevant to said location and where to find them in the map room. The software also links to a site where a heatmap will be maintained. The client for this project is the Colorado School of Mines library and maps room, specifically Chris Thiry the maps and GIS librarian. They wish to improve the experience of people visiting the library by telling them about maps relevant to their hometown. They also want to gather demographics about where people visiting the library are from. There are no previous versions of this software, and we will be starting everything ourselves. The Colorado School of Mines library and maps room have a database of the maps they have available. This database contains information on types of content and where to find it. The information is currently in two Excel spreadsheets which list maps and either US county or country respectively. The hardware interface will consist of a Windows computer where visitors can interact with the maps with a mouse. The computer was provided by the library, and we did not need to set it up, but we did need to optimize the software for the display size. We only used a few acronyms that we define here and in the appendix, the first being GIS which stands for Geographic Information System. We

also used some APIs which stand for Application Programming Interface and UML which stands for Unified Modeling Language. Users of the software will be visitors to the Colorado School of Mines library and maps room. The Colorado School of Mines library and Chris Thiry are the clients and stakeholders because they will be using the software to collect demographic information on visitors and can improve their collections based on those demographics. The client also wanted the software to be open source for other institutions to use, making other libraries, museums, and academic institutions stakeholders too. The client requested that the software be able to work without a major overhaul for three years, after which someone will look at what needs to be updated. The client will need to be able to maintain the maps database for that information to be helpful to users. Chris Thiry will also need to go to the computer to send himself the visitor data and use that data to update the heatmap we link to.

II. Functional Requirements

The functional requirements of this project revolve around delivering the end outcome of displaying the location of maps within the library to the end user using our GUI. The user needed to be presented with a home screen containing three map options. These include maps of Colorado, the United States, and the world. The user needed to be able to select one of these options which presents them with a bound map where they can then place a pin on their hometown. The Colorado and United States maps are broken down by state and county and the World map will be broken down by country. The end user also needed to be able to zoom in and out on all maps. After the pin is placed, the country, county, state, and time stamp are stored in our visitor schema which can be referenced by the library later to determine where visitors are from. Finally, the shelf location of where the end user can find a map from the hometown needed to be displayed on the screen in a pop-up that is easy to read and derive from.

III. Non-Functional Requirements

The non-functional requirements of this project include the end-user experience and aesthetic of the application. Our client made the point that the application should look “cool”. The GUI should be smooth and easy to use, which encourages more people to use it. The software will not collect any information or data from users outside of what is required for successful functionality. The software should be reliable and intuitive enough for people to use it without instructions. Our client has additionally requested that the

software be open source so that other libraries across the country would be able to use it. Furthermore, the software should use open-source resources to minimize software development costs and optimally run on the provided hardware.

IV. Risks

We identified four major risks to our project that could slow down our development. The first risk we found was that there might be difficulty communicating the technical details of our project with a client who is not super technical. We decided that it was likely we would run across this risk at some point during the project, but that the impact of this would be small as the solution is easy. To help with this problem we made sure to communicate everything clearly with the client and ensured he understood all the technical aspects that he needed to. The second risk that we found was an unfamiliarity with the necessary software to get this project working from scratch. We classified this risk as very likely as none of us had worked with all the necessary software and we all needed to learn some. We decided that this risk would have a moderate impact on our project because any time we came across something that we didn't know, all members of the group took time to follow tutorials and do research to familiarize ourselves. The third potential risk we identified was the library database being messy and hard to use, which we classified as moderately likely. This would have meant taking time to clean the database, we classified this as having a moderate impact on our project. There was no real way to mitigate this risk, but once we did receive the database, we only had to do some cleaning, as the database was reasonably clean already. The final risk we identified was running into difficulties making the code open source, which we classified as likely to happen. This would only have a minor impact if we ran into this, since the code was built of components that can all be open source already, and we just needed to make sure we gave good documentation as to how another institution could change the code to fit their system.

V. Definition of Done

The definition of done consists of the main idea that the application at least meets the aspect of a minimum viable product. The minimum useful features include displaying maps with pinch and zoom functionality, displaying a pin for visitor's hometown when they click on a map, processing the data from the end user click, and generating a pop-up consisting of the shelf location where a map from a clicked location can be found in the library. Additionally, the team will test the software extensively to ensure that the data

populated is accurate and locations are returned as expected. The database will also be checked for accuracy. The product will be delivered as a Windows application designed to work on a computer in the library at the end of the semester.

VI. System Architecture

Figure 1 displays our application design and flow. Our application starts on a home screen that uses Java Swing to display action buttons and information. The most important buttons are the Colorado state, United States, and world buttons, clicking on any of these buttons redirects the user to the map screen but with different bounds to zoom in on the selected local. Then the user clicks the map in the area of their hometown, a digital pin appears, and the program calls another file with API information. The APIs convert the latitude and longitude data collected by the user's click into the county, state, and country data to be collected and displayed to the user. If the user had selected an invalid area, such as the ocean or Antarctica, the API returns an error, and the program informs the user and asks them to try again. After the API returns usable data, a python script is called to get the library shelf data and return it to the main java program. The returned shelf data is displayed in a pop-up and the user can then return to the map and select another location or return to the home screen.

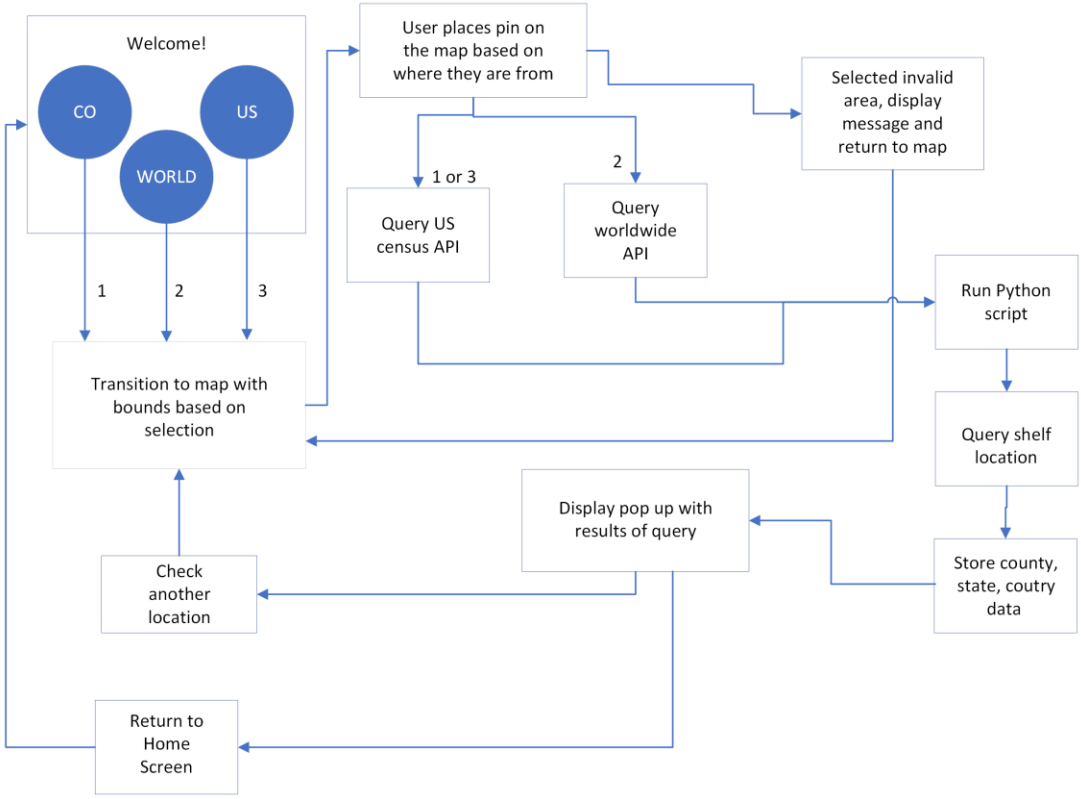

Figure 1 – Final Application Flow

In implementing our software design, we encountered a few challenges with the map display and the usability of the data. As mentioned in our risk assessment, none of the team members were familiar with maps software or APIs, therefore getting the map displayed was the first big design challenge we addressed. The other hurdle we faced was the usability of the data returned from a user clicking on the map. Any map API that we would attempt to use would return values for latitude and longitude, while the library database contains information on the county, state, and country. Creating a new database to convert the data would have been a massive undertaking, so we used a couple more APIs to convert the data.

The central feature of our program is the map display. We considered a few map APIs when starting the project, however, our lack of budget and time constraints were heavy factors in the decision of which API to use. In choosing an API we heavily considered what we needed our map to do. The client wanted a map that would zoom and pan, this is called a slippy map and is similar in style to Google Maps. The Google Maps platform is universally recognized and has an API system; however, the numerous options and varying price points led us to consider other options first. Open Street Map and Map Box both had free-tier APIs but also steep learning curves.

A technical feature of this scope that none of the team members are familiar with was going to take time, but an important part of working with limited time is to know when you have been attempting something too long and when you should start another approach. Our initial program utilized ArcGIS as it was free and utilized JavaFX for a cleaner GUI. We found that it required an extensive installation of SDK modules that were difficult to install and get working. In addition, this method made it difficult to pass data the way we needed to in order to provide the functionality requested by our client. The documentation and example of loading an OpenStreetMap layer using ArcGIS Enterprise SDK can be found here [1]. After a couple of weeks of trying to get the Open Street Map and ArcGIS APIs integrated into the Java project, we took another look at Google Maps and ended up achieving the map display by using a static Google Maps API and the jxMaps library.

Another large feature of our project is converting the latitude and longitude data that we return into state and county data if the click was in the US or country data otherwise. For this, we needed a reverse geocoding API, which takes latitude and longitude data and returns information, usually in a JSON object, about the location of that point. For the state and county conversions inside the US, we decided to use the FCC's reverse geocoding API [2], because it is based on direct census data from the 2020 census and would most likely have the most up-to-date information. That API is updated every census, so if our code is still in

use in 2030, then all it would take is changing one line to update it to the most recent data. For the country conversion, there was not one agency that provided an up-to-date API, so we decided to go with the Nomanatim reverse geocoding API from OpenStreetMap [3]. This API does an accurate job of converting each point to its corresponding country but is not regularly updated with any border changes. This API also does not have all the country names in the JSON object that is returned, therefore we decided to return a 2-digit country code instead since the API has all those in the JSON object. After a user clicks, we first run the point through the Nomanatim API, which tells us the country code that the point is in, and if the code is US, then we send the point through the FCC's API to get the state and county, if it is not we only return the country code. This data is all stored in a Java class, which can then be accessed to send into the python script.

VII. Technical Design

Our program's main functions and GUI display are written in Java, however, Python's pandas library was the most straightforward method of accessing our databases. To accommodate the Python script in a Java project, we utilized a process builder. The integration of a Python script in a Java project also necessitates that the python files be added to the source packages when building the project into an executable. The technical design of the application had the goal of creating an executable that combines all aspects of user interface efficiency while providing the most functionality in a local setting. The specific parts we desired to highlight are covered below.

The most important aspect of our technical design revolved around the implementation of our databases. This is essential in the processes of returning correct results to the user and collecting information to be utilized later in aspects such as a heat map or improving library map collections based on geographic trends. We used two databases in our project, one from which we retrieved data from and one which stored user data. Figure 2 is a schema for both of our databases, which includes 4 tables, Visitor, Library Inventory, US Counties, and Countries. The Visitor schema contains all the information from a visitor to the library who clicks on the map. It has sections for their country, state, county, and a timestamp of when they clicked. Whenever a user uses our map, they will be stored in this visitor schema, which populates in an excel file that Chris Thiry can access easily from the machine. The US Counties schema contains a lot of information from Mr. Thiry that we do not use, the main things we use are state, county name, and drawer case number. We can use the countries schema, similarly, using just country name, drawer case number, and

country code. These can be joined into one library inventory schema that joins on the shelf locations and holds all the information on the location. When a click is converted to country, state, and county by the reverse geocoding APIs, we can then query the appropriate database, either US Counties or Countries, and return the associated map shelf location for that click.

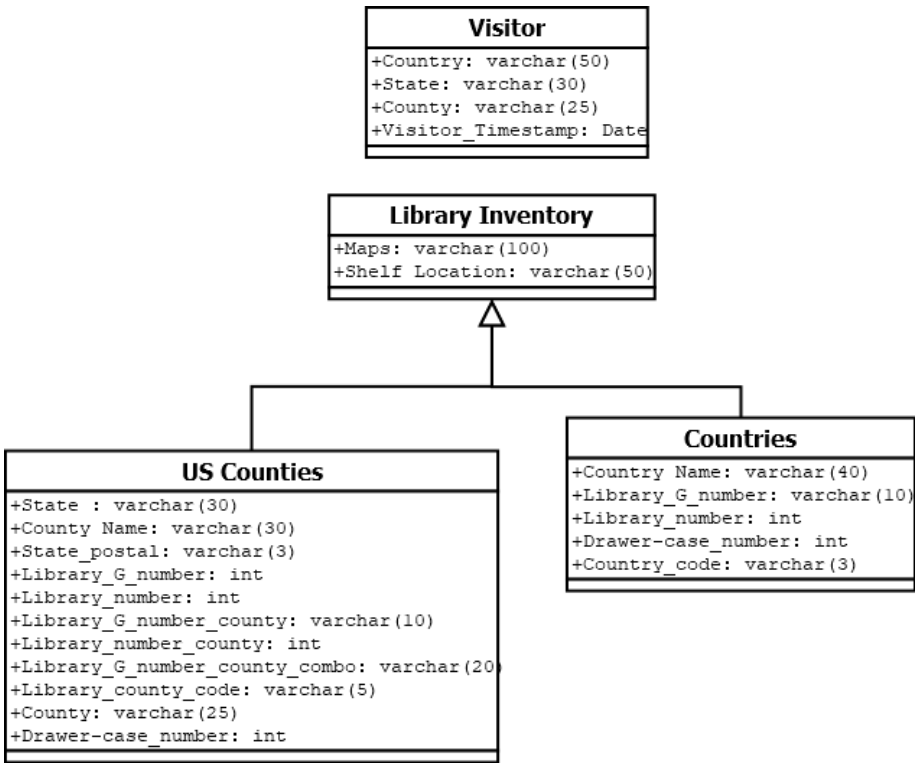

Figure 2 – Database Schema

One of the techniques we used to create the functionality of returning the shelf location of the maps from our local schemas was externally running a python script within our Java project to utilize data processing from libraries such as pandas. Figure 3 below shows the code used to call the Python script via a process builder. Figure 4 below demonstrates how the process builder works.

```
ProcessBuilder builder = new ProcessBuilder( command: "python",  command: "Field_Session/build/classes/generate_shelf_locs.py",
        command: click.country,  command: click.state,  command: click.county);
Process process = builder.start();
BufferedReader stdInput = new BufferedReader(new InputStreamReader( in: process.getInputStream()));
```

Figure 3 – Process Builder Code

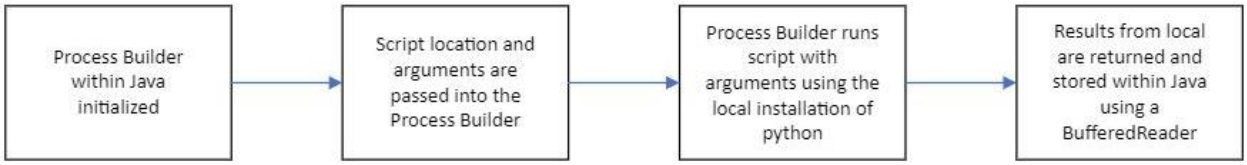

Figure 4 – Process Builder Flow

Initially, a process builder within our Java is initialized where “each ProcessBuilder instance manages a collection of process attributes [4].” The process builder takes in a command specifying the language to be used in the local execution. In our case, this is python as we are utilizing pandas' data functionality to process shelf locations associated with a user map click. Furthermore, the script location and arguments are passed into the process builder. This allows us to call our script locally and pass in attributes including country, state, and county. The builder then runs the script with arguments using the local installation of python. This means that the local installation of python must be fully functional with all the relevant library installations for execution to properly occur. Lastly, results need to be returned from the local run of the external script for values to be passed down further through the application. In our case, we were able to use a buffered reader to get the input stream from the console and store it. ProcessBuilder does have built-in redirect input and redirect output methods that allow for input and output sources to be specified in other use cases [4].

The central part of the program is the map display. To achieve the map display we used a static Google Maps API and the jxMaps library. Figure 5 is how the map displays after the user clicks the United States button on the home screen. The jxMapViewer library and API is an “open-source Swing component created by the developers at SwingLabs [5].” The library was released in around 2010 but was revived into jxMapViewer2, an updated open-source version [6].

Figure 5 – United States Map Display

Additionally, while slightly less technical, we utilized jxMapViewer's waypoint feature to meet our client's requirement of a pin drop appearing on the map when a user clicks a location. Figure 6 contains code demonstrating the waypoint functionality. Here we use a HashSet of waypoints and a painter tool to create the map overlay where the pin displays appropriately. Additionally, we utilize features such as Geo Positioning to set map bounds and other functionality.

```
// this function displays the waypoints
public void addWaypoint(GeoPosition location) {
    //create a set of waypoints
    Set<Waypoint> waypoints = new HashSet<Waypoint>();
    // if returning to the map after pop up, null is passed to this function and waypoints will be erased
    if (location != null) {
        waypoints.add(new DefaultWaypoint( coord: location));
    }
    //create a WaypointPainter to draw the points
    WaypointPainter painter = new WaypointPainter();
    painter.setWaypoints(waypoints);
    jxMaps.setOverlayPainter( overlay: painter);
}
```

Figure 6 – Waypoint Code

VIII. Software Test and Quality

To ensure that our code meets all quality standards, we made sure to test 5 specific aspects of our code. The first, most basic level of testing we did was to make sure that the basic functionality meets the definition of done. We must test that clicking all over the map returns the correct outputs of resources in the library, these should be accurate in most cases but could be weird in areas with border disputes. We tested along borders and coastlines and tested clicking in unpopulated areas such as oceans or Antarctica as well. The program returned an accurate location and the correct resources for a valid location or an error message for unpopulated regions. The second category of testing we did was having unit tests for user input and heat map outputs. We tested that the heat map is generated correctly based on the stored data of user locations, and we entered some dummy data to test this. We want the heat map to be accurate for the locations of people, and link correctly every time. For edge cases, we tested high and low numbers of people in regions, made sure the heat map adjusts to this and returns the correct map. The third major category of testing we did is verifying that we didn't go over any API call limits. We ended up using all free APIs for our software so there is nothing we need to check here. The fourth category of testing was to make sure that no latitude or longitude data is stored permanently. We made sure that with a click we retrieved latitude and longitude, then translated them to generic data about county, state, or country, but we deleted latitude and longitude

after the conversion. We also won't collect any other personally identifying information from any users. The last category of testing that we did is testing to accommodate many users in one day. We tested with large quantities of dummy data to make sure that everything continues to update as we go. We only needed to test with one at a time, since there will only be one computer, and only one person can use it at a time. We needed to test the same number of people that would potentially be visiting a museum in a day since this might be implemented in one. Following these 5 categories of testing helped to ensure that our code is high quality and well-developed.

IX. Project Ethical Considerations

Looking at the ACM ethics guidelines, we found six that apply to our project [7]. The first is ACM 1.3 which is to be honest and trustworthy. We wanted to ensure that we are honest with how data is stored from users, and we made sure that we only use the data for the purposes that people are expecting. The next guidelines are ACM 1.6 and 1.7, which state to respect the privacy and confidentiality of our users. We do not collect any personal identifying information and specific click information, (latitude and longitude) is only stored temporarily. The only stored information is county, state, and country, while latitude and longitude data is destroyed when the user returns to the home screen. Our fourth guideline is ACM 2.1 which says we should strive to achieve high quality in both the processes and products of professional work. We worked to make our product the highest quality possible and ensure that our client was happy with the final product. ACM 3.1 states that the public good should be the central concern during all professional computing work. We followed this in our project by making sure that our project is designed to help the user and that the priority behind our motivation is helping visitors to the library. The last principle we needed to follow is ACM 4.1 which states that we needed to uphold, promote, and respect the principles of the code. These line up closely with some of the IEEE ethical guidelines, so we decided to only list the ACM guidelines [8]. We followed this in making our project open source, making sure that any future users of the code also follow our principles, and that the code could be maintained by future developers. In following these ethical principles throughout development, it was unlikely that we would run into any ethical challenges, and our final product would be a demonstration of ethical computing.

X. Results

Our final product includes most initial features at an acceptable level approved by our client. These include implementing a home screen with three map selection options of Colorado, the United States, and the world. Each one of these selections correctly displays with appropriate bounds. The map has optimal click, zoom, and panning functionality. In addition, the user can drop a waypoint marker on the map that returns a pop-up consisting of shelf locations where the user can find maps of that location. The user can then choose to return to the map and choose another location or return to the home screen. From the home screen, the user can additionally select an option that will take them to a browser consisting of a heat map that the client will upload periodically.

The database implemented consists of a country, US-counties, and visitors' schemas that interact with the application. Upon each map click, both country and us-county schemas are queried given a conversion input from our API latitude and longitude parser. The visitor's schema is populated after each waypoint drop with data needed to populate the heat map. The database should remain functional for the lifetime of the application and certainly for three years of operation without intervention as requested by the client.

The only feature that we did not implement was automatic heatmap generation from stored data. After extensive attempts, we could not create the map and insert it into the application dynamically for the user to view. We pivoted with guidance from our client to have an optional browser option that they will periodically update it.

Testing was performed extensively throughout the development process. We tested border regions and disputed areas to ensure that results were accurate and returned appropriately. Additionally, oceans and Antarctica do not return values and suggest choosing another location. Time was also a large factor in testing, and we found that some regions can take a few seconds to generate results but overall is relatively fast.

Usability tests demonstrated proficiency in the user's ability to successfully use the application and yield the desired result. The returned data accurately allows the user to locate the shelf location within the library of maps that have relevance to their hometown or a clicked location. This greatly enhances the visitor experience given the experience of locating these at faster rates with more convenience offered than a manual search.

XI. Future Work

The baseline product has been produced with most of the functionality requested by the client. Future work for the project would mostly include complete integration of a heatmap that populates automatically and is presented to the user given data collected within local schemas. Additionally, future work would include aesthetic changes to enhance the appeal to the user. We would also suggest that the project be modified and loaded onto a touch screen for usability. We believe a touchscreen option would increase user traffic and create a better user experience.

Resources required to further work on the project include NetBeans, Python3 (version 3.9 or higher), the latest java installation, and a computer or laptop running Windows OS. Additionally, the local installation of python (environment variables and pip installs) must be functioning for the application to successfully operate.

Knowledge & skills required include familiarity with jswing for GUI development, proficiency in both Python and Java languages, an understanding of databases and structure, and external scripting (I.e. running a python script within a java application and returning results). Knowledge of using and integrating APIs and open-source libraries is also essential for the further development of our application.

We estimate that the recommendations we have made for further work would require a timeline of roughly a month. Most of the time would be spent working on the full integration of the heatmap feature and the rest would be allocated toward graphic design improvements.

Given more time and resources, we would have liked to implement the heat map as mentioned above from the collected data in the visitor schema. We found that it requires a lot of new development to achieve this, which is the main reason we were not able to achieve the heat map feature. Future work could be done by anyone who uses our open-source code, and they could implement any of these features by working off our base code and knowing the necessary software.

XII. Lessons Learned

Throughout this project, we learned a lot about the challenges of starting a project from scratch and bringing it to the final implementation. The first major thing that we learned is that APIs can be difficult to implement in Java, but once they are working, they are very useful for handling small tasks for the code. Another thing we learned is that map GUIs are difficult to get working and getting the GUI to work with the specific map API took many attempts before it started to work. Another challenge we encountered and learned from was integrating python into a java program. We learned a lot about how to integrate one language into another, and we faced some struggles getting the code to work correctly with the integration. Once we got it working it was super useful, as we were able to do all our database work in python which is much nicer.

One main lesson that we learned from all these struggles is that building a program from scratch requires multiple iterations and multiple frameworks until we can find the correct set of pieces that can all run together in one IDE. Another lesson that we learned from doing this project was that developing the application flow is incredibly important to implementing the code. Without our helpful flow diagrams, it would have been much harder to know how all the parts communicate with each other, and the implementation was much easier when we had a plan.

XIII. Team Profile

Jordan Ehrlich: Senior in Computer Science. From Eagle, Colorado. Internship experience in python & SQL development where I have scripted datafile generation and automated oil & gas well inventory with a current focus on report streamlining.

Wes Golliher: Senior double major in Data Science and Statistics. From Longmont, Colorado. Internship experience developing and integrating data science models with quota attainment and sales performance data in python at Xactly in Denver.

Clare Garski: Senior in Computer Science, minor in Mathematics. From Colorado Springs, Colorado. Internships working in data flow, automation, virtual labs, cybersecurity, and networking. Work experience at campus Information Technology and Solutions.

References

[1] esri, “ArcGIS Documentation,” *ArcGIS*. [Online]. Available: https://doc.arcgis.com/en/. [Accessed: 29-Nov-2022].

[2] FCC, “Federal Communications Commission,” *FCC Area API*. [Online]. Available: https://geo.fcc.gov/api/census/. [Accessed: 29-Nov-2022].

[3] Nominatim, “Nominatim Reverse Geocoding API,” *Nominatim Demo*. [Online]. Available: https://nominatim.openstreetmap.org/ui/reverse.html. [Accessed: 29-Nov-2022].

[4]“Class ProcessBuilder,” *ProcessBuilder (Java Platform SE 7)*, 24-Jun-2020. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html. [Accessed: 29-Nov-2022].

[5] “Building maps into your swing application with the jxmapviewer: Java.net,” *Building Maps into Your Swing Application with the JXMapViewer | Java.net*, 2010. [Online]. Available: https://web.archive.org/web/20100309081606/http://today.java.net/pub/a/today/2007/10/30/building-maps-into-swing-app-with-jxmapviewer.html. [Accessed: 29-Nov-2022].

[6] Msteiger, “MSTEIGER/Jxmapviewer2: Jxmapviewer2,” *GitHub*. [Online]. Available: https://github.com/msteiger/jxmapviewer2. [Accessed: 29-Nov-2022].

[7] ACM, “The code affirms an obligation of computing professionals to use their skills for the benefit of society.,” *Code of Ethics*. [Online]. Available: https://www.acm.org/code-of-ethics. [Accessed: 29-Nov-2022].

[8] IEEE, “IEEE code of Ethics,” *IEEE*. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 29-Nov-2022].

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
API	*Application Programming Interface – allows out program to communicate with another computer program.*
GIS	*Geographic Information System – a database containing geographic information.*
UML	*Unified Modeling Language – used to create a diagram to better understand our database system.*
GUI	*Graphical User Interface – the system the user interacts with to change the display or get information.*
SDK	*Software development kit – a collection of software development tools.*
Open source	*Denoting the software as free and available to the public.*