HawkEye CS 411 Final Project

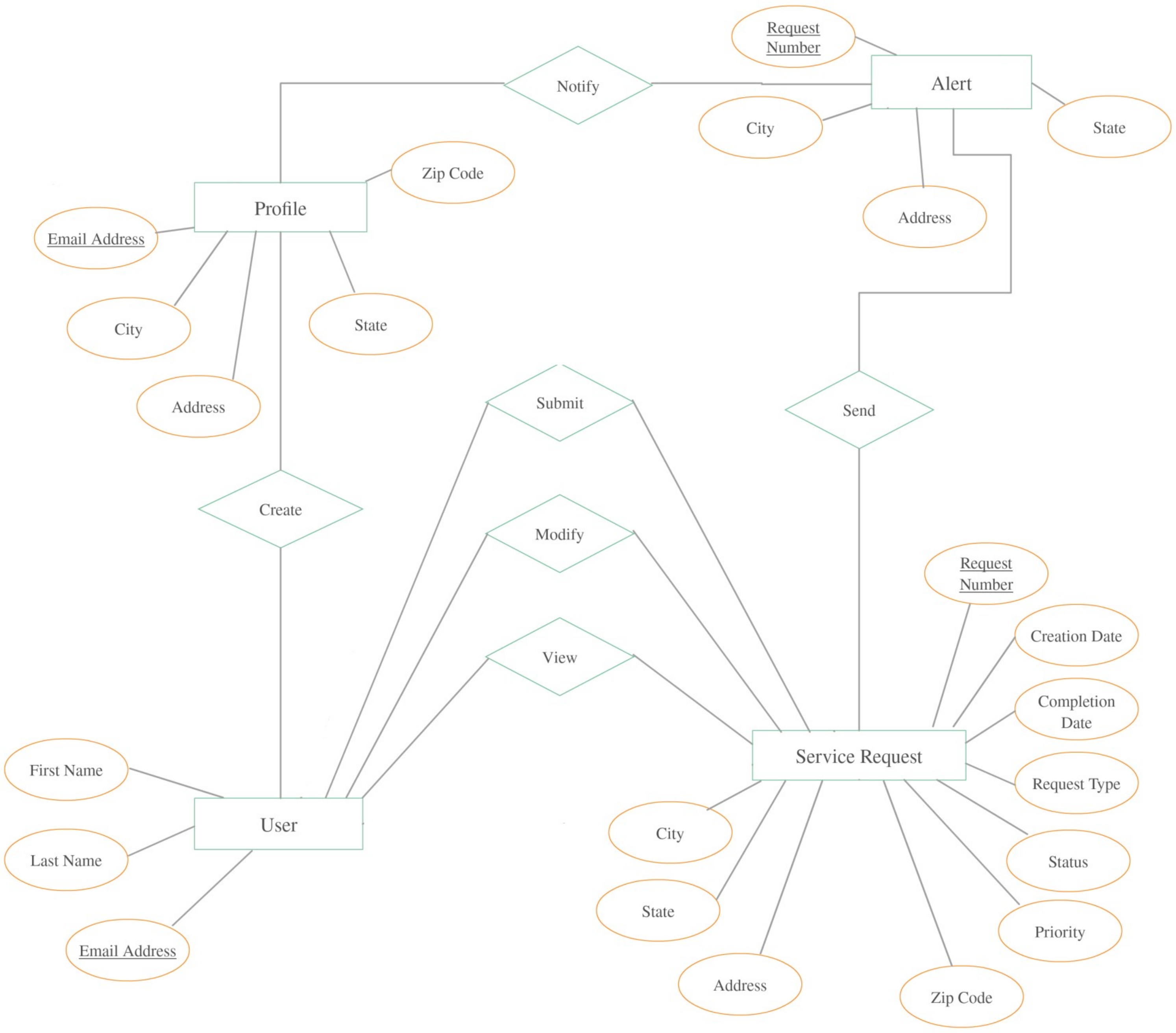
Team Name	HawkEye
Members and Names (NetIDs)	Alice Yu (aliceyu2), Collin Ryals (cryals2), Ahtesham Ali (saali4)
Emails	aliceyu2@illinois.edu, cryals2@illinois.edu, saali4@illinois.edu
Team Captain	Alice Yu (aliceyu2)
Project Title	Chicago Service Requests (CSR)
Project Summary	Using a dataset from Kaggle containing records of the different service requests created in the years 2018 and 2019, our database application will provide Chicago residents the ability to report new requests, check the status of existing requests, and be alerted of newly created or completed service requests that have been reported in their area.
Project Description	● **Application description**: Our application will take all 311 Chicago city service requests that were created in the years 2018-2019 and perform different basic and advanced functions on the database to create a user-friendly interface that will help residents track the status of different types of service requests based on user input. CSR will help residents report new requests, view all existing requests, and stay informed about all newly created and completed service requests in their area via an autonomous notification system. ● **Usefulness**: In contrast to other applications that currently only allow users to check the status of different types of service requests (e.g., https://www.chicago.gov/city/en/depts/311.html), our application will have the ability to display the 20 highest priority service requests that were created within a one mile radius of any user-specified location. In addition, our application will allow the user to opt-in to an autonomous alert system that will notify the user if any new requests have been opened in their area or if the status of any of those nearby service requests

	have changed. ● **Dataset description**: The dataset comes from a precompiled database from Kaggle called “Chicago 311 Service Requests”. Our main dataset consists of records resulting from a filtering of the original precompiled database to show only the service requests created in 2018 and 2019 involving pot holes. ● **Functionality description**: Our application’s basic functions will include operations such as insertions, updates, deletions, and searches and will be used to clean up older service requests that have been marked as completed over 2 years ago from the database, create additional requests, update existing requests, and display only subset of the database based on a specified service request type, the status of a service request, and the user-specified location. Our application’s advanced functions will involve the implementation of geofencing and an automated email alert system. Geofencing is a feature that uses a device’s GPS or longitude/latitude coordinates to define geographical boundaries. It will allow our application to translate an address provided by a user to geographical coordinates that our project can then use to create a geographical boundary around the specified address. The technical challenge with geofencing comes with being able to quickly ensure that the address combination provided by a user is valid and if not, prevent the user from creating the request with that address and ask for a valid address to be given instead. This means that within milliseconds of submitting an address, our application should be able to run through our geofencing code and immediately deduce whether or not the provided address is valid. Our automated email alert system then leverages our geofencing capability to then ensure that only users within 5 miles of a service request’s location will receive an email alerting them of the request’s existence. Due to the fact that everything is handled on the backend, our application has the ability notify users real-time without the need for manual intervention. The technical challenge with an automated email alert system comes with being able to check whether or not existing users have a location specified in their profile and ensuring that only users meeting all requirements will be able to have our application create and send an email alert via a SMTP server that we had to learn how to leverage

	and configure, all within a couple of seconds maximum. Since we do not want to spam an user's inbox we have defined our requirements as thus: alerts only go to a user if their profile address is within 5 miles of a request's specified address, alerts get sent to that user only if they did not create the request themselves, and an alert only gets sent if the request is new. ● **Advanced techniques**: Our application will also leverage more advanced techniques such as transactions, triggers, compound statements, constraints, and views.
ER Design Statements	● A user can only create 1 profile ● Each profile can be created by only 1 user ● A user can submit multiple service requests ● A user can make many service requests ● Service requests can be made by many users ● Service requests must contain a unique request number, a status, a creation date, a completion date, a priority number, city, state, zip code, address and a request type ● Service requests can send multiple alerts ● Alerts can be sent by multiple service requests ● Each service request must contain an unique request number that does not exist anywhere else ● Accounts can be notified of multiple alerts ● Alerts can notify multiple accounts ● Alerts will contain the following information: a service request number, a service request type, a status, and a location ● A profile must contain a city, state, zip code, and address ● Each user has an unique email address, a first name, a last name, and a password ● Users can have a combination of a first and last name that are not necessarily unique ● Users can view multiple service requests ● Service requests can be viewed by multiple users ● Users can modify multiple service requests ● Service requests can be modified by multiple users ● A profile will contain the user's associated email address
Relational Schema	● Profile(P<u>rofile.emailAddress</u>, city, state, zipCode, address) ● User(U<u>ser.emailAddress</u>, firstName, lastName, password, Profile.emailAddress) ● ServiceRequest(S<u>erviceRequest.requestNumber</u>,

	creationDate, completionDate, requestType, status, priority, city, state, zipCode, address) ● Alert(Alert.requestNumber, city, state, address) ● Notify(Profile.emailAddress, Alert.requestNumber) ● Send(Alert.requestNumber, ServiceRequest.requestNumber, city, state, zip code, address) ● Submit(User.emailAddress, ServiceRequest.requestNumber) ● Modify(authorities_User.emailAddress, status_ServiceRequest.requestNumber) ● View(User.emailAddress, ServiceRequest.requestNumber)
Development Plan	● **Database choice:** MySQL ● **Software platforms:** Django, Apache ● **Software languages:** JavaScript, Python ● **How and where to find data:** Our data will come from a subset of records that originally exists in a precompiled Kaggle database called "Chicago 311 Service Requests". The subset of records that our database will use will come from a filtering of the original database to show only the service requests created in 2018 and 2019 involving pot hole service requests.

ER Diagram


Request Number
Alert
Notify
City
State
Zip Code
Profile
Address
Email Address
City
State
Address
Submit
Send
Create
Modify
Request Number
Creation Date
View
Completion Date
Service Request
First Name
Request Type
User
City
Last Name
Status
State
Priority
Email Address
Address
Zip Code

Development Plan - Project Timeline and Labor Division

Milestone	Due Date	Milestone Description	Labor Division
1	2/24/19	a. Install all necessary tools, packages, and libraries onto the VMs b. Download all software platforms and languages (if they do not already exist) onto the VMs c. Install and set up the Apache web server onto the VMs d. Set up the backend by creating all of the necessary database tables and populating those tables with the correct columns e. Inject data from the dataset into the database	● Alice: a-d ● Cole: a-e ● Ahtesham: a-c
2	3/3/19	a. Create a FQDN and research how to keep the backend reachable by the front-end at all times b. Create the service request status web page for the website c. Implement a search function that leverages backend operations and reflects the results on the front-end	● Alice: b, c ● Cole: a ● Ahtesham: b
3	3/10/19	a. Design each web page for the web site b. Implement the front-end homepage design for the website c. Implement the login web page for the website d. Implement the registration web page for the website	● Alice: a, b ● Cole: a, d ● Ahtesham: a, c
4	3/17/19	a. Implement the profile web page for the website b. Implement the service request creation web page for the website c. Implement the service request home web page for the website d. Add basic functionality to website (CRUD)	● Alice: a, c, d ● Cole: b, d ● Ahtesham: c, d
5	3/24/19	a. Begin implementing the trigger-based push notification feature (advanced feature #1) b. Begin implementing the geo-fencing feature (advanced feature #2) c. Finalize the decision about which advanced techniques to implement	● Alice: a-c ● Cole: a-c ● Ahtesham: a-c

6	3/31/19	a. Create a demo video that demonstrates CRUD operations (initial video) b. Finish the trigger-based push notification feature (advanced feature #1) c. Finish the geo-fencing feature (advanced feature #2)	● Alice: c ● Cole: b ● Ahtesham: a
7	4/7/19	a. Upload the demo video demonstrating CRUD operations (initial video) b. Implement 1 advanced technique c. Implement 1 advanced technique d. Implement 1 advanced technique	● Alice: a, b ● Cole: c ● Ahtesham: d
8	4/14/19	a. Implement 1 advanced technique b. Implement 1 advanced technique c. Add additional GUI features for better visual appeal	● Alice: a ● Cole: b ● Ahtesham: c
9	4/21/19	a. Perform unit and automation tests b. Add additional GUI features for better visual appeal c. Create a demo video that demonstrates advanced technique operations (final demo)	● Alice: b, c ● Cole: b, c ● Ahtesham: a, c
10	4/28/19	a. Upload the demo video demonstrating advanced function and technique operations (final demo)	● Alice: a

Link to the system's search page:
http://sp19-cs411-12.cs.illinois.edu:8000/serviceRequest/
Note: You must have an account on the website in order to see this page.
Example Login Information: user1, user1Password

Link to the initial demo video:
https://drive.google.com/file/d/1-czzY8bMwHTamUgcISEp1-Yf9CMdXPCb/view?usp=sharing

Link to the final demo video:
https://drive.google.com/file/d/1vZ3FITmi3d8D6U-CYY4AqPtWSsttZdMe/view?usp=sharing
https://mediaspace.illinois.edu/media/t/1_amsfunoy

Link to the source code:
https://github.com/aliceyu2/CS411-CSR/tree/finalDemo