BACnet – Subjective Chat: Nicknames

Introduction to Internet and Security

Delia Jacobs, Neha Selvan

Eingereicht: 18.07.2021

Einführung

Wir erweiterten den *Subjective Chat* mit dem Feature *Nickname*. Bei diesem Feature handelt es sich um die Möglichkeit, als Person A einer Person B einen *Nickname* zu geben, falls wir im *Subjective Chat* einen gemeinsamen Chat erstellt haben. Bei Person A wird daraufhin der *Nickname* als neuer Name für die Person B im Privatchat sowie in Gruppenchats angezeigt. Person B wird über den neuen *Nickname* in Kenntnis gesetzt. Dieser wird in einer Liste mit aller *Nicknames*, die Person B erhalten hat, erscheinen. Person A kann außerdem ihren eigenen *Username* verändern. Dabei erfahren alle ihre Kontakte von dieser Änderung, damit deren Chats mit Person A unter ihrem neuen Namen angezeigt wird. Ein weiteres Feature unserer Implementation ist, dass Person A eine Liste erstellen kann mit Namen, die sie nicht zugewiesen bekommen möchte. Dadurch wird es ihren Kontakten nicht möglich sein ihr ein *Nickname* zu vergeben, indem einer der unerwünschten Namen vorkommt.

Die wichtigsten Fragen, die wir und zu Beginn stellen mussten, waren wie die Kommunikation im *Subjective Chat* zum ersten Mal aufgebaut wird und wie der Austausch zwischen zwei Partnern abläuft. Den auf diesen Abläufen aufbauend, würden all unsere Funktionen beruhen.

Hintergrund

In unserem Projekt wollten wir ursprünglich die *BACnet ++* App erweitern. Nach einer erfolglosen, intensiven Installations- und Integrationsphase mit *AndroidStudio* und *Pycharm* wurde uns jedoch geraten auf den *Subjective Chat* zu wechseln.

Im *Subjective Chat* gibt es keine Profile für Kontakte wie dies in weit verbreiteten Chatapplikationen der Fall ist. Stattdessen wird nur der Chat und der Name des Kontakts auf dem *Userinterface* dargestellt. Darum haben wir uns entschieden alle Funktionen, die die Namen des *Users* oder dessen Kontakte betreffen, in einem eigenen Fenster anzuzeigen.

Um Informationen auszutauschen, wird zunächst ein *Event* erstellt und dem entsprechenden *Feed* hinzugefügt. Dieser bestimmt an wen das *Event* verschickt wird. Beim Erstellen eines *Events* werden die Metadaten (wie zum Beispiel die *Feed-ID* und die Sequenznummer dieses *Events*), die *Signatur* und der Inhalt, der verschickt werde soll, benötigt. Die Funktion eines *Events* kann beliebig definiert werden. Alle zur Abarbeitung des *Events* benötigten Informationen, werden in einem *Dictionary* mitgeliefert. Die Abarbeitung, also die Reaktion auf ein erhaltenes *Event,* wird in der Klasse *event_handler* durchgeführt. Die Funktion eines *Events* kann beliebig definiert werden. Um die Art eines *Events* zu spezifizieren, wird die Applikation, in der das *Event* gebraucht wird und der *Eventname* überliefert. Die beiden Informationen werden mit einem ‚/' abgetrennt. Dies sieht zum Beispiel so aus: *‚chat/sendname'*

Das aller erste *Event* wird in der Klasse *feed_control* erstellt. Hierbei wird der *Masterfeed* des *Users* mit dem Namen *„Anon"* erstellt. Alle anderen *Events* werden dem erstellten *Masterfeed* zugeordnet. Das erste *Event* eines neuen Chats wird in der Klasse *subjective_chat* erstellt und beinhaltet den eigenen *Username* und die Art des Chats (Privat- oder Gruppenchat) als zu übertragenden Inhalt.

Implementation

Als Grundlage haben wir den bestehenden Code des *Subjective Chats* von vorherigen Gruppen verwendet. Die Implementation unseres *Features* haben wir in diese Grundlage hineingeflochten. Dabei haben wir vor allem die Klassen *subejctive_chat.py*, *event_handler.py* und *uiFunctionsHandler.py* angepasst. Wir haben zusätzliche Methoden hinzugefügt und bestehende erweitert. Wir waren stets darauf bedacht mit unseren Änderungen das bestehende Projekt nicht zu zerstören.

Das grundlegende Prinzip, dass wir für die meisten unserer Funktionen verwendet haben, ist das Speichern der geänderten Namen (oder anderer nötiger Information) in externen Dateien.
Wir haben dieses Prinzip aus zwei Gründen angewandt. Der eine Grund ist, dass wir die Informationen über die verschiedenen Namen längerfristig speichern müssen. Sie müssen auch erhalten bleiben bei mehrmaligem Öffnen und Schliessen des Chats, des *Sneakernets* oder von *Feed Control*.
Das zweite Problem war schwieriger zu lösen. Die, durch ein oben beschriebenes *Event* erhalteneen Informationen, müssen in aller Regel auch für die Klasse *subjective_chat* zu Verfügung stehen. Von dort aus hat man aber keinen direkten Zugriff.
Die geeignetste Lösung für diese beiden Probleme ist, wie gesagt, das Verwenden von externen Dateien.

Bei unserer ersten Funktion, das Ändern des eigenen *Usernames*, wird der Name im *username.pkl* File angepasst. Das File speichert den *Username* und den dazugehörigen *Key* des Logins. Beim Ändern des eigenen Namens wird dies beim Schlüssel *'Username'* im File entsprechend angepasst. Für jede andere Stelle im Code, in der diese Information gebraucht wird, hat man nun durch dieses File Zugriff. Der vorherige *Username* geht verloren.

Das zweite Feature ist das Vergeben oder Ändern eines *Nicknames* einer Person, mit der man mindestens einen gemeinsamen Chat hat. Dazu ist ein weiteres File notwendig, wir haben es *connectedPerson.pkl* genannt. Darin wird ein *Dictionary* mit den *Usernames* derPersonen als Schlüssel und den *Nicknames* als Werte gespeichert. Hat eine Person noch keinen *Nickname*, wird der *Username* stattdessen als Wert gespeichert.
Beim Ändern eines *Nicknames* wird der Wert entsprechend angepasst. Sollte eine der, mit Person A verbundenen Personen ihren *Username* anpassen, so wird auch der Schlüssel in meinem *connectePerson.pkl* File angepasst. Von dieser Änderung erfährt Person A durch das entsprechende *Event*. Damit Person A nicht aus Versehen zwei ihrer Kontakte derselbe *Nickname* zugeordnet wird, wird dieser vor dem Zuweisen des *Nicknames* überprüft. Es wird geschaut, ob der *Nickname* im *connectePerson.pkl* File als Wert bereits aufgeführt ist. Ist dies der Fall, so wird der *User* mit einer Infobox benachrichtigt und der *Nickname* wird nicht akzeptiert. Ist der *Nickname* noch nicht vergeben, so kann er verwendet werden.

Die dritte Funktion ist das Anzeigen aller einer Person A, von anderen Personen, zugewiesener *Nicknames*.
Dies funktioniert wiederum nach demselben Prinzip wie bisher. Sobald jemand Person A einen *Nickname* zuweist, wird ein *Event* verschickt anhand dessen Person A weiss, welcher *Nickname* ihr zugeteilt wurde. Die Namen werden im *my_names.txt* File gespeichert, wodurch sie ihr später im

Chat angezeigt werden können. Diese Liste hat vor allem einen Zweck: man hat die Kontrolle darüber, welche Namen einem zugewiesen werden dürfen und welche nicht. Dies geschieht durch unser viertes *Feature*, die *Unwanted Names*.

Person A kann also eine Liste mit Namen, die sie stören, womöglich weil sie beleidigend sind, anlegen.
Wenn ihr nun ein Name zugewiesen wird, so erfährt sie von dem neuen Namen durch ein *Event*. Im *Event Handler* wird geprüft, ob der Name akzeptable ist, also nicht auf der Liste steht oder, ob er inakzeptable ist. In diesem Fall werden folgende Informationen in das *resetName.txt* File geschrieben: Zuerst *True*, um später zu wissen, dass der Name inakzeptable war, dann folgt der Name, der neu zugwiesen wurde, darauf folgt der Name, der zuvor zugewiesen war (zum Beispiel ein anderer *Nickname*) und zum Schluss steht, von welchem *User* der neue Name kam. Alle Informationen sind jeweils durch ein '/' abgetrennt.
Wenn Person A im *Subjective Chat* den *Names-Tab* öffnet, wird geprüft, ob im *resetName* File *True* und die notwendigen Informationen enthalten sind. Wenn ja, wird ein neues *Event* kreiert, welches beim entsprechenden *User* zur Folge hat, dass der *Nickname* wieder auf den ursprünglichen *Username* geändert wird.
In das *resetName* File wird der *Default Value: False*, wieder hineingeschrieben.

Um Probleme zu vermeiden, muss vor dem ersten Updaten im *Sneakernet* das Login in den *Subjective Chat* erfolgen. Ansonsten kommt es zu Komplikationen mit dem erstellen von gewissen Files.

Resultate und Diskussion

Der *Subjective Chat* sieht, mit den neuen Features, wie folgt aus:

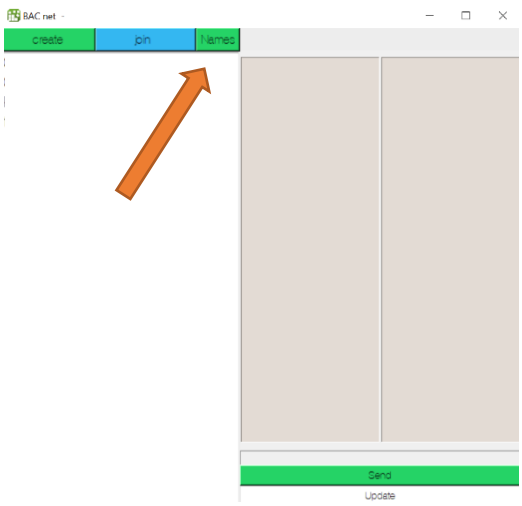

Abb. 1 Chat-Window mit Names-button

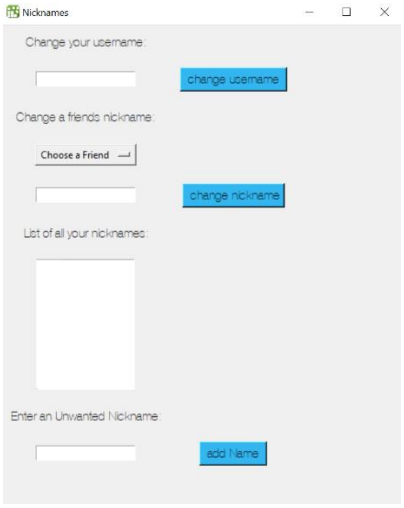

Abb2. Nicknames-Window

Links ist der *Subjective Chat* zu sehen, mit dem neuen *Names-Button* in der Menu Leiste.
Rechts ist das neue *Nicknames-Window* abgebildet. Die vier, oben beschriebenen, Funktionen sind hier von oben nach unten angezeigt. Es enthält alle geplanten Features sowie die ungeplante Erweiterung, *Unwanted Nicknames*.

Unser allgemein grösstes Problem war das Speichern, Aktualisieren und Verwalten von Namen. Dazu mussten wir einige Dateien neu anlegen, was oft zu Verwirrungen geführt hat über Inhalt und Darstellung der Daten. Nach einiger Zeit und ein paar verlorenen Stunden, erstellten wir ein detailliertes Protokoll zu Inhalt und Darstellung der Daten, womit es im weiteren Verlauf zu keinen Problemen mehr kam.
Genauso eine Verwirrung mit den Dateien führte dazu, dass auf einmal der gesamte Chat nicht mehr funktionierte. Auch *Feed Control* und *Sneakernet* waren davon betroffen. Wir verloren viel Zeit mit der Suche nach der Ursache und noch einmal einiges an Zeit beim Beheben des Fehlers.
Danach hatten wir einen deutlich besseren Überblick über das gesamte Projekt und die darauffolgenden Arbeiten waren wesentlich leichter zu implementieren.

Die zweite grosse Hürde hatten wir bei den *Unwanted Names* zu überwinden. In unserem Konzept, wäre notwendig gewesen, ein neues *Event* im *Event Handler* selbst zu erstellen. Dies war allerdings nicht möglich, da wir dafür weiter Klassen hätten importieren müssen. Diese *Imports* führten allerdings zu zyklischen *Imports*, weshalb wir diesen Plan nicht realisieren konnten.
Daraufhin mussten wir einen gänzlich neuen Ansatz, wiederum über externe Files, finden, welcher schlussendlich zum Ziel führte.

Während dem Testen unserer Features stiessen wir auf Probleme beim Verbinden mit mehreren Personen. Beim *Trusten* in der *Feed Control* Anwendung erhielten wir immer denselben Error, nämlich, dass die Bezeichnung des neuen *Feeds* gleich lautet wie bereits beim vorherigen. Wir hatten den Verdacht, dass dieser Fehler nicht an unserer Implementation lag, sondern bereits vorher so auftrat. Wir wandten dasselbe Vorgehen auf den ursprünglichen Code, ohne unsere Änderungen, an, und konnten denselben Fehler reproduzieren. Daraus schlossen wir, dass es tatsächlich nicht an unserer Implementation lag. Um die Funktionalität unserer Features dennoch testen zu können, erstellten wir eine neue Klasse *Test.py*, mit der wir künstlich mehrere Personen hinzufügen konnten. Damit waren wir in der Lage unsere Features vollumfänglich zu testen.

Bei der Vorbereitung auf unser Projekt ist uns aufgefallen, dass es beim *Subjective Chat, Feed Control* und dem *Sneakernet* jeweils eine *Change Username* Funktion gab. Daher hatten wir die Idee diese drei Funktionen zu vereinheitlichen. Bei genauerem Hinsehen wurde jedoch deutlich, dass die Funktionen nicht zusammenfügbar sind. Bei *Feed Control* ist der *Username* völlig unabhängig vom *Username* im *Subjective Chat*. Eine Vereinheitlichung hätte hier keinen Effekt gehabt.
Im *Sneakernet* ist die Funktion *Change Username* zwar als *Button* vorhanden, jedoch ist sie nicht implementiert. Hier macht es grundsätzlich keinen Sinn den *Username* mit jenem vom *Subjective Chat* zu koppeln, den bei jedem Login wird ein neuer *User* erstellt. Dessen Namen hat keinen Zusammenhang mit jenem vom *Subjective Chat*. Es kann also ein beliebiger Name eingegeben werden, was überhaupt kein Einfluss auf das korrekte Updaten der Feeds hat. Da der *User* nach dem Beenden der Anwendung wieder verworfen wird, wäre es wohl sinnvoller das Login mit dem *Username* komplett weg zu lassen.

Konklusion

Wie bei jedem Projekt sind einige Startschwierigkeiten wohl unumgänglich. Nachdem wir von der *BACnet++* App zum *Subjective Chat* wechseln mussten, war unser gesamtes Konzept hinfällig und wir mussten uns einen völlig neuen Plan zurechtlegen. Dies kostete uns viel Zeit.
Als eher schwierig stellte sich das Einbauen unseres Codes in den bereits bestehenden Code. Wir haben daraus gelernt, dass man mit Vorsicht vorgehen muss, um nichts zu zerstören, was bereits funktioniert hat.
Für das restliche Projekt mussten wir flexibel bleiben und oft von unserer ursprünglichen Idee abweichen. Daraus haben wir gelernt, dass es immer mehr als einen Weg gibt und es sich lohnt ein Problem manchmal von einer völlig anderen Seite zu betrachten.