|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sterowniki PLC** | | | | | | |
| Rok akademicki | Termin | Rodzaj studiów | Kierunek | Prowadzący | Grupa | Sekcja |
| 2017/2018 | *Czwartek* | SSI | Informatyka | dr inż. Michał Maćkowski | PSI | 4 |
| *16:15 – 17:45* |
| Ocena: |  | Komentarz: |  | | | |

Dokumentacja projektu

Data wykonania: 2018*-07-11*

Temat projektu:

System dostępu do pomieszczeń

Link do repozytorium: github.com/PSI-2018/SPLC-2018

|  |
| --- |
| Skład sekcji: |
| Łukasz Błasiak |
| Katarzyna Urbasik |
| Jakub Czachor |
|  |
|  |

1. **Treść oraz założenia projektowe**

Celem projektu jest opracowanie i wykonanie systemu pozwalającego na realizację autoryzowanego dostępu do pomieszczeń przy użyciu telefonów komórkowych. Zamkiem do pomieszczenia steruje stale włączony komputer PC z dostępem do Sieci (przyjęto, że jest wyposażony w kartę przekaźników sterujących zamkami w drzwiach). Otwarcie może zostać dokonane przez odpowiednie polecenie wysłane przez TCP/IP do komputera sterującego zamkami.

Należy opracować dwa programy:

1. Dla komputera PC: odbieranie przez TCP/IP żądań otwarcia zamków, weryfikacja żądań i otwieranie zamków (fakt "otwarcia może być tylko sygnalizowany na ekranie; rzeczywista aplikacja wymagałaby zastosowania np. przekaźników sterowanych przez USB). Powinna istnieć możliwość definiowania listy uprawnionych użytkowników zamka (z prostą edycją - może być plik tekstowy ze spisem użytkowników ich danymi). Program powinien logować akcje użytkowników;
2. Dla telefonów użytkowników zamka (można przyjąć, że będzie to aplikacja dla jednego, wybranego systemu). Aplikacja powinna na żądanie wysyłać odpowiedni (konfigurowalny - z indywidualnym hasłem) pakiet TCP/IP do komputera PC.
3. **Serwer TCP/IP – specyfikacja funkcjonalna**

Główną częścią projektu jest serwer TCP/IP obsługujący żądania od klientów, sterujący dostępem do pomieszczeń oraz komunikujący się z baza danych. Całość została zaimplementowana w języku obiektowym Java z wykorzystaniem pakietu Java.net do obsługi komunikacji sieciowej.

Do prawidłowej pracy serwera, w pierwszej kolejności należy poprawnie skonfigurować plik *config.properties* znajdujący się folderze *cfg* w głównym katalogu projektu. Zawiera on podstawowe ustawienia konfiguracyjne takie jak m.in. adres IP urządzenia na którym pracuje serwer oraz port. W przypadku braku lub niepoprawnie wypełnionego pliku konfiguracyjnego, aplikacja poinformuje o tym fakcie stosownym komunikatem.

Serwer został zaimplementowany z wykorzystaniem wielowątkowości, co pozwala na obsługę każdego klienta w osobnym wątku. Powyższe rozwiązanie skraca kolejkę oczekiwania na obsługę żądania. Dodatkowo, ze względu na niską cykliczność zapytań, na każde żądanie tworzone jest pojedyncze połączenie, zamykane zaraz po jego obsłużeniu. Redukuje to liczbę otwartych połączeń do liczby aktualnych żądań, a nie liczby uruchomionych aplikacji (które mogą być aktywne w tle).

Na potrzeby projektu zaimplementowany został protokół komunikacyjny, znajdujący się w klasie *Protocol*. Posiada on m.in. listę dozwolonych komend nadesłanych przez klienta. Odpowiada on głównie za walidację żądania (poprawność polecenia oraz odpowiednia liczba argumentów) oraz oddzielenie argumentów do odpowiedniego kontenera. W protokole zostały zaimplementowane odpowiednie gettery umożliwiające odczytanie powyższych wartości. Serwer obsługuje polecenia służące do logowania, otwierania drzwi oraz zwracania listy pomieszczeń w formacie JSON. Poniżej znajduje się tabela zawierająca opis poszczególnych poleceń:

|  |  |  |
| --- | --- | --- |
| **POLECENIE** | **ARGUMENTY** | **ZWRACANA WARTOŚĆ** |
| OPEN | <email> <hasło> <nr\_pokoju> | OPENED ROOM NR: … |
| NO PRIVILEGES TO OPEN ROOM NR: … |
| AUTHORIZATION FAILED |
| WRONG ROOM NUMBER |
| WRONG COMMAND FORMAT |
| LOGIN | <email> <hasło> | LOGGED |
| AUTHORIZATION FAILED |
| WRONG COMMAND FORMAT |
| GET\_DOORS\_LIST | <email> <hasło> | <JSON zawierający listę pomieszczeń> |
| AUTHORIZATION FAILED |
| WRONG COMMAND FORMAT |

Do poprawnej pracy wymagana jest również baza danych MySQL wraz z poprawną konfiguracją. Do obsługi komunikacji z bazą danych, wykorzystany został pakiet Java.sql oraz oficjalny MySQL connector w wersji 5.1.46. W pliku konfiguracyjnym serwera muszą zostać podane takie ustawienia jak: adres do bazy wraz z portem, nazwa użytkownika oraz hasło. W przypadku błędnej konfiguracji serwer zasygnalizuje powyższy fakt stosownym komunikatem.

W bazie znajdują się dwie tabele: *user* oraz *privilege*. Pierwsza przechowuje informacje o użytkownikach takie jak id, e-mail, hasło oraz dwie flagi: ze stanem aktywności konta oraz czy jest to konto z uprawnieniami administratora. Druga tabela przechowuje natomiast informację związane z dostępem do poszczególnych pomieszczeń: id uprawnienia, numer pokoju oraz id użytkownika, który posiada dane uprawnienie.

W serwerze został dodatkowo zaimplementowany logger archiwizujący wszystkie aktywności zachodzące po stronie serwera. Pojedynczy wpis związany z zapytaniem przyjmuje następujący format:

<data> <godzina> <email> <adres IP> <rezultat polecenia>

Logger zapisuje również połączenia i rozłączenia klienta z serwerem. Pełną historię logów można znaleźć w pliku *logs.log* znajdującym się folderze *cfg* w głównym katalogu projektu. Dodatkowo każda aktywność jest wypisywana na konsoli serwera.

Dodatkowo w kodzie źródłowym, do każdej metody oraz pola zostały zamieszczone odpowiednio sformatowane komentarze. Z wykorzystaniem narzędzia *Javadoc,* umożliwiają one generację dokumentacji w postaci strony WWW na podstawie zamieszczonych w kodzie źródłowym znaczników w komentarzach.

1. **Aplikacja mobilna**
   1. Specyfikacja wewnętrzna
      1. *ClientCommunication*

Jest to klasa zawierająca protokół komunikacyjny klienta, umożliwia wymianę informacji z częścią serwerową.

Posiada konstruktor z parametrami umożliwiającymi przekazanie portu i hosta oraz metody umożliwiające komunikację:

* *checkLogin(userEmail, userPassword)* – metoda umożliwiająca sprawdzenie poprawności logowania użytkownika o adresie email *userEmail* oraz haśle *userPassword* do systemu
* *getDoorList(mail,password)* – metoda umożliwiająca odebranie listy dostępnych pokoi od serwera
* *checkPrivileges(userEmail, userPassword, roomNumber)* – metoda umożliwiająca sprawdzenie czy użytkownik (*userEmail, userPassword)* ma prawa dostępu do danego pokoju (o numerze *roomNumber).*

Za komunikację z serwerem odpowiada klasa *AsyncAction*, która zawiera metodę *doInBackground* oraz metody będące odpowiednikami wyżej opisanych metod.

* + 1. *Floor*

Klasa Floor zawiera listę obiektów typu Room i jest odpowiednikiem pojedynczego piętra.

Poza bezparametrowym konstruktorem, zawiera metody:

* *setFloorName(newName)* – metoda służąca do ustawiania nazwy piętra
* *getFloorName(), getAllRooms(), getAllRoomsNames() ­­*– gettery do uzyskania nazwy piętra, listy pokoi oraz nazw pokoi danego piętra
* *addRoom(roomName) –* metoda umożliwiająca dodanie kolejnego pomieszczenia do piętra
  + 1. *LoggedActivity*

Aktywność widoczna po zalogowaniu się użytkownika do systemu. Posiada metody:

* *onCreate(savedInstanceState) –* metoda zarządzająca aktywnością
* *manageRooms(position)* - metoda umożliwiająca wyświetlenie odpowiednich pokoi
* *getRoomsFromServer() –* metoda służąca do odczytania przesłanej przez serwer listy pokoi oraz dodanie ich do odpowiednich list
* *getFromSpinners()* – metoda umożliwiająca odczytanie wartości wybranych przez użytkownika w spinnerach (numeru pokoju i piętra)
  + 1. *MainActivity*

Główna aktywność aplikacji. Zawiera między innymi metody:

* *onCreate(savedInstanceState)* – metoda zarządzająca działaniem aktywności
* *checkId(mailGiven,passwordGiven) –* metoda umożliwiająca użytkownikowi zalogowanie się (sprawdzenie poprawności podanych danych, wyświetlenie odpowiedniego komunikatu, ewentualne umożliwienie przejścia do kolejnej aktywności)
* *moveOn()* – metoda umożliwiająca przejście do aktywności *LoggedActivity*
  + 1. *Room*

Obiekty tej klasy są odpowiednikami pojedynczych pokoi danego piętra. Klasa zawiera jedynie dwie metody: *setNumber(newRoom)* oraz *getNumber()*, będące odpowiednio setterem i getterem numeru danego pokoju.

* + 1. *Util*

Klasa umożliwiająca dostęp do danych zapisanych w pliku .properties. Jedyną metodą jest *getProperty(key, context)*, która zwraca odczytane dane.

* 1. Specyfikacja zewnętrzna

Po włączeniu aplikacji, użytkownik natrafia na ekran z napisem „Log In” oraz dwoma polami, służącymi odpowiednio do wpisania adresu email oraz hasła użytkownika. Użytkownik powinien uzupełnić te dane oraz zatwierdzić, klikając w przycisk „LOGIN”.

Jeżeli któreś z pól nie zostanie uzupełnione, pojawi się napis „Fill both fields”.

Jeżeli podane dane będą nieprawdziwe, pojawi się napis „Wrong credentials”. Jeżeli użytkownik poda poprawne dane, widniejące w bazie, pojawi się napis „Logged” oraz zostanie otwarta kolejna aktywność. Na ekranie pojawią się (patrząc od góry do dołu): napis „Pick a floor”, rozwijana lista pięter, napis „Pick a room”, rozwijana lista pokoi w ramach wybranego piętra, przycisk „OPEN” oraz przycisk „LOG OUT”. Użytkownik powinien kolejno wybrać piętro, wybrać pokój i nacisnąć przycisk „OPEN”. Jeżeli użytkownik posiada prawa do otwierania wybranego pokoju, pod przyciskiem „OPEN” pojawi się napis „Room [numer pokoju] was succesfully opened”. W przeciwnym wypadku pojawi się napis „You have no rights to influence this room!”. W dowolnym momencie użytkownik może nacisnąć przycisk „LOG OUT”, co będzie skutkowało wylogowaniem i powrotem do pierwszego okna aplikacji.

1. **Panel administracyjny**
   1. Specyfikacja wewnętrzna
      1. *Log*

Klasa ta odpowiada za wyświetlenie okna logowania do panelu administracyjnego. Wykorzystując funkcje klasy ManageDB nawiązywane jest połaczenie z bazą danych sprawdzające poprawność wpisanych danych (email i hasło) oraz czy osoba logująca się ma odpowiednie uprawnienia (jest administratorem). Jeśli logowanie powiedzie się, okno zostaje usunięte metodą dispose(), a w jego miejsce wywoływany jest panel administracyjny (poprzez stworzenie instancji klasy CardLayoutApplication. Okno logowania to tak naprawdę jeden JFrame, w którego skład wchodzi JPanel, JTextField, JPasswordField oraz JButton.

* + 1. *CardLayoutApplication*

Jest to klasa zawierająca implementację niektórych metod interfejsu ItemListener, służąca do reprezentacji panelu administratora z wykorzystaniem architektury CardLayout. Architektura CardLayout jest wkorzystywana najczęściwej w aplikacjach typu wizzard, gdzie zamiast tworzyć nowy JFrame co kolejną zatwierdzoną akcję, wykorzystuje się jeden JFrame wypełniany kolejnymi panelami, co przekłada się na czytelniejszy interfejs użytkownika (przykładem takiej aplikacji jest Adobe Photoshop z jednym oknem zawierającym obszar roboczy oraz narzędzia, w przeciwieństwie do aplikacji Gimp posiadającej osobno obszar roboczy i osobno narzędzia). JFrame utworzony z wykorzystaniem tej klasy posiada jeden ListBox, na którego zmianę stanu reaguje aplikacja, zmieniając wyświetlany obecnie JPanel. Oba obiekty JPanel posiadają po dwa obiekty JButton oraz JTextField. W celu skrócenia czasu prezentacji, wpisywany tekst będący hasłem nie jest ukrywany z wykorzystaniem JpasswordField (aby uprościć prezentację walidacji wpisywanych haseł).

* itemStateChanged(ItemEvent evt) – implementacja metody interfejsu ItemListener, wykrywająca akcje użytkownika w obrębie panelu.
* addComponentToPane(Container pane) – metoda odpowiadająca za inicjalizację i odpowiednie rozmieszczenie obiektów w obrębie panelu i zapisująca oba (panele) w kontenerze przechowującym je jako tzw. karty.
  + 1. *ManageDB*

Klasa ManageDB udostępnia metody do podejmowania akcji związanych z dodawaniem, usuwaniem i walidacją wysyłanych i zwracanych danych. Ponadto służy także do nawiązywania połączenia z bazą danych i wykrywania potencjalnych nieprawidłowości.

* postUser(String email, String password) – metoda służąca do dodania nowego użytkownika do bazy danych I ewentualnego usunięcia istniejącego, jeśli ma ten sam email. Metoda zwraca true jeśli operacja dodawania zakończyła się pomyślnie lub false w przeciwnym wypadku.
* postPrivilege(String id, String room) – metoda służąca do dodania nowego uprawnienia do bazy danych I ewentualnego usunięcia istniejącego, jeśli ma to samo UserId. Metoda zwraca true jeśli operacja dodawania zakończyła się pomyślnie lub false w przeciwnym wypadku.
* deleteUser(String email, String password) – metoda służąca do usunięcia istniejącego użytkownika z bazy danych. Metoda zwraca true jeśli operacja usuwania zakończyła się pomyślnie lub false w przeciwnym wypadku. Jeśli do id użytkownika przypisane są jakieś uprawnienia, są one automatycznie usuwane (kaskadowo) przez bazę danych.
* deletePrivilege(String id, String room) – metoda służąca do usunięcia istniejącego uprawnienia z bazy danych. Metoda zwraca true jeśli operacja usuwania zakończyła się pomyślnie lub false w przeciwnym wypadku.
* getConnection() – metoda służąca do nawiązywania połączenia z bazą danych. Jeśli połączenie z bazą danych się powiedzie, jest ono zwracane przez metodę, w przeciwnym wypadku zwracany jest null.
* checkIfAdminSignIn(String email, String password) – metoda służąca do weryfikacji danych logowania do panelu administracyjnego. Jeśli użytkownik nie istnieje, ma niepoprawne dane lub nie jest administratorem to metoda zwróci null, w przeciwnym wypadku metoda zwróci true.
  + 1. *DataValidator*

Klasa udostępniająca metody do walidowania danych.

* isValidEmail(String email) – metoda służąca do sprawdzania poprawności wpisanego maila. Jeśli mail odpowiada kryteriom (wykorzystany regex) to zwróci true, w przeciwnym wypadku false.

* 1. Specyfikacja zewnętrzna

Po uruchomieniu aplikacji, na ekranie zostaje wyświetlone okno „Login Autentification”, posiadające dwa pola do wpisywania tekstu oraz przycisk „Log in”. Pierwsze pole tekstowe służy do wpisania adresu email, drugie do hasła użytkownika. Jeśli wpisany email nie spełnia kryteriów określonych przez aplikację, użytkownik zostanie o tym poinformowany nowym oknem z wiadomością, po którego zamknięciu użytkownik będzie miał szansę zalogować się ponownie. Analogiczne poinformuje użytkownika aplikacja, jesli ten wpisze niepoprawne dane, lub nie jest administratorem.

Po przejściu procesu autentyfikacji, okno logowania zamknie się samoistnie, a na ekranie pojawią się dwa okna – panel administratora oraz przysłaniająca je wiadomość powitalna. Po zamknięciu wiadomości powitalnej, administrator może dodawać lub usuwać użytkowników, a także nadawać i odbierać im uprawnienia korzystając z rozwijanego pola z wartościami „User” oraz „Privilege”. Domyślną wartością jest pole „User”, które udostępnia administratorowi panel do zarządzania użytkownikami. Aby modyfikować uprawnienia, należy wybrać z rozwijanego pola wartość „Privilege”, wtedy aplikacja wykryje tę zmianę i załaduje panel do zarządzania uprawnieniami. W ten sposób administrator może przełączać się między funkcjami aplikacji z uwagi na operacje, jakie chce wykonać na bazie danych.

Wpisywane przez administratora dane są sprawdzane przez aplikację pod kątem poprawności, a następnie podejmowane są odpowiednie kroki, aby wykonać operację lub poinformować administratora o przyczynie niepowodzenia.

Po zakończeniu pracy, wystarczającym jest zamknięcie okna panelu administracyjnego.

1. **Wnioski**

Podczas realizacji projektu, wykorzystany został język programowania Java wraz z podejściem obiektowym. W celu zdalnego sterowania pomieszczeniami, został zaimplementowany dedykowany serwer obsługujący przychodzące polecenia. Pozwoliło to na dokładne zrozumienie modelu TCP/IP oraz biblioteki programistycznej Java*.net*.

W celu zdalnego kontrolowania pomieszczeniami, utworzony został dedykowany klient dla systemu operacyjnego Android, komunikujący się z serwerem TCP/IP. Podczas realizacji aplikacji, poznaliśmy szczegółowiej cykl życia aplikacji, zdarzenia asynchroniczne, programowanie wielowątkowe oraz obsługę zdarzeń.

Wartym uwagi jest również baza danych, która przechowuje informacje odnośnie użytkowników wraz z ich uprawnieniami. Do jej realizacji wykorzystany został wolnodostępny system zarządzania relacyjnymi bazami danych – MySQL. Podczas jej tworzenia, rozwinęliśmy umiejętności z zakresu projektowania i tworzenia tabel oraz odpowiednich relacji między nimi.

W celu zarządzania uprawnieniami, utworzony został panel administracyjny. Pozwala on na modyfikację uprawnień poszczególnych użytkowników z wykorzystaniem interfejsu graficznego. Powyższe rozwiązanie eliminuje konieczność ręcznego modyfikowania bazy danych przez administratora oraz zwiększa czytelność znajdujących się w niej danych.

Dodatkowo programy były testowane wielokrotnie zarówno ze względu na poprawność ich działania, jak i efektywność algorytmów. Testowanie z różnymi kombinacjami argumentów, ich ilościami oraz wartościami przebiegło zgodnie z założeniami   
i oczekiwaniami. Program dobrze pobierał dane wprowadzone przez użytkownika oraz w razie braku ich poprawności, informował użytkownika stosownym komunikatem.