Dokumentacja

Projekt grupowy

20@KIBI Google Glass jako inteligentny prompter

Członkowie zespołu:

Leszek Buława

Karol Kawalerczyk

Maria Kucułyma

Tomasz Scharmach

Opiekunowie projektu:

dr inż. Jacek Rumiński

Sylwester Gryzio - Kainos

Pamela Pieglowska - Kainos

Tomasz Świderek - Kainos

Spis treści

Wstęp

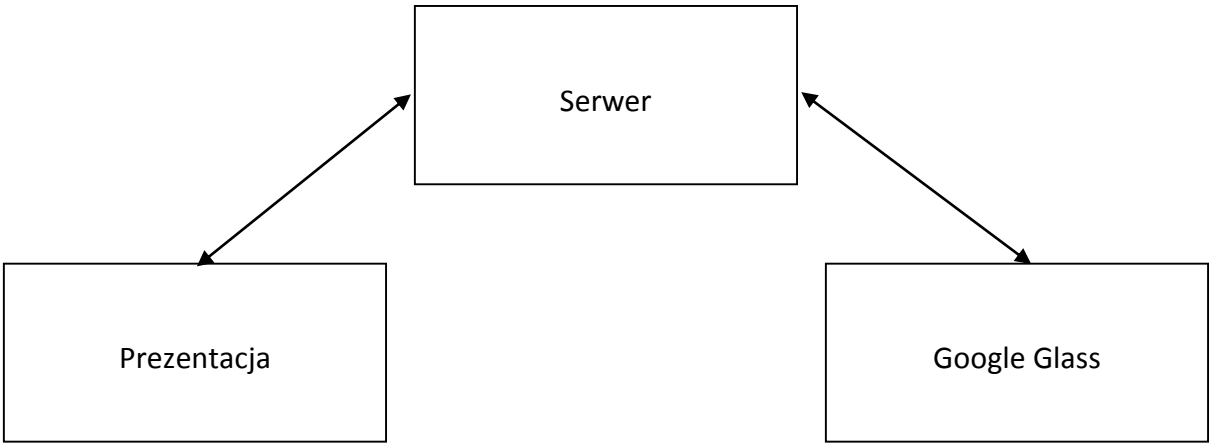
Realizowany przez nas projekt ma na celu wspomaganie prelegentów w trakcie prowadzenia prezentacji. Wielokrotnie w przygotowaniu prezentacji mają swój nieoceniony udział notatki do poszczególnych slajdów. Zazwyczaj zapisywane są one na kartce papieru lub znajdują się na ekranie komputer, na którym uruchamiana jest prezentacja. W jednym przypadku musimy stale coś trzymać w ręce, w drugim nie możemy swobodnie poruszać się po sali, w której prowadzimy prelekcje, gdyż musimy znajdować się przy komputerze w celu odczytania notatek. Nasz projekt jest odpowiedzią na ten problem. Na urządzenie mobilne marki Google, jakim jest Google Glass stworzyliśmy aplikację, przy pomocy której jesteśmy w stanie sterować przejściami między slajdami, kontrolować czas prezentacji oraz odczytywać notatki bez potrzeby wykonywania dodatkowych czynności. Projekt został zlecony przez firmę Kainos oraz nadzorowany przez nią przez cały okres prac.

Założenia projektu

- Stworzenie serwera, który będzie pośredniczył między prezentacją, a Google Glass.
- Stworzenie aplikacji klienckiej obsługiwanej z poziomu urządzenia mobilnego.
- Realizacja połączenia „okulary-serwer-prezentacja”.
- Wyświetlanie notatek do poszczególnych slajdów na ekranie okularów.
- Dobór czcionki z poziomu aplikacji serwerowej.
- Kontrolowanie czasu prezentacji poprzez wyświetlanie timer’a na ekranie okularów.

Sposób rozwiązania problemu

Poniższy schemat ilustruje działanie naszego projektu:



Zasada działania projektu jest następująca:

1) Klient uruchamia aplikację serwerową i startuje serwer.
2) Następnie uruchamia aplikację kliencką z poziomu Google Glass.
3) Aplikacja kliencka wyszukuje dostępne serwery wysyłając zapytanie broadcastowe.
4) Serwer wysyła odpowiedź wraz ze swoim IP.
5) Nawiązywane jest połączenie TCP pomiędzy okularami, a serwerem.
6) Klient z poziomu aplikacji serwerowej otwiera prezentację.
7) Serwer pobiera notatki zawarte w prezentacji.
8) Serwer wysyła notatki do urządzenia mobilnego.
9) Sterowanie prezentacją odbywa się na zasadzie wymiany danych pomiędzy aplikacją kliencką, aplikacją serwerową, do której załadowana jest prezentacja.

Aplikacja serwerowa

Klasa Main.java

Jest to główna klasa aplikacji serwerowej. W tej klasie tworzona jest metoda *MainFrame*, która zawiera interfejs graficzny programu, oraz ustawiana jest jako widoczna.

Klasa MainFrame.java

Klasa ta odpowiada za utworzenie GUI (Graphic User Interface). Zawarte w niej są poszczególne metody:

initFrame() – tworzy ramki do interfejsu, wywołuje dalej opisane metody: *createButtons(), createLabels()* oraz *initStartSettings().*

createButtons() – metoda odpowiedzialna za utworzenie przycisków w GUI pozwalające na wystartowanie serwera, zatrzymanie go, załadowanie prezentacji, otwarcie logów (oraz wyczyszczenie ich), wybór czcionki, uruchomienie timer'a (oraz resetowanie go), minimalizację aplikacji do systemtray'a oraz wyjście z aplikacji. Znajduje się w niej także metoda i*temStateChanged()*, która monitoruje status timer'a i przesyła dane o jego statusie do innych metod oraz aplikacji klienckiej.

createLabels() – metoda odpowiadająca za utworzenie etykiet informujących o stanie aplikacji serwerowej, takich jak: status serwera, nazwa komputera, adres IP komputera, czy typ połączenia z aplikacją kliencką.

initStartSettings() – metoda inicjująca ustawienia początkowe aplikacji, czyli zaraz po uruchomieniu. Wywołuje metodę *setIpAndHostName()*, która uzyskuje dane o nazwie komputera oraz jego adresie IP.

actionPerformed() – metoda monitorująca działania użytkownika w zakresie GUI i wywołująca odpowiednie metody, po wybraniu odpowiedniego przycisku.

startServer() – metoda aktywuje się po wciśnięciu przycisku, uruchamia osobny wątek tworzący serwer, z którym później łączy się aplikacja kliencka.

stopServer() – aktywacja metody przyciskiem, zatrzymuje działanie serwera oraz zatrzymuje wątek uruchomiony w metodzie *startServer()*.

minimizeFrame() – minimalizuje aplikację serwerową do systemtray'a, po akcji użytkownika.

exitServer() – zamyka okienko aplikacji serwerowej.

openLog() – tworzy dziennik zdarzeń w postaci pliku notatnika, gdzie są zapisane działania na aplikacji.

clearLog() – czyści dziennik zdarzeń.

manual() – otwiera podrzędny interfejs, w którym ręcznie można załadować prezentację, odczytać ilość slajdów oraz notatki w niej zawarte.

openPresentation() – metoda ładująca prezentację do aplikacji oraz wczytująca ilość slajdów oraz notatki przy pomocy metod zawartych w klasie *Extractor.java*.

getCurrentFont() – zwraca aktywny wybór czcionki.

clickButton() – metoda odpowiadająca za wybór czcionki, włączenie oraz reset timer'a.

setIpAndHostname() – metoda korzysta z klasy *LocalHost.java*, wczytuje adres IP i nazwę komputera, a następnie wyświetla je w interfejsie.

changeLabel() – metoda odpowiada za zmianę wyświetlanego statusu serwera.

isTimerEnabled() – metoda sprawdzająca checkbox „Enable Timer" oraz zwracająca wartość *true/false* w zależności od wyniku.

Klasa ManualFrame.java

Klasa odpowiedzialna za stworzenie podrzędnego menu wywoływanego z poziomu menu głównego. Za jego pomocą możemy ręcznie załadować prezentację, odczytać ilość slajdów oraz notatek. Metody w niej zawarte to:

initFrame() - tworzy ramki do interfejsu, wywołuje dalej opisane metody: *createButtons(), createLabels()* oraz *initStartSettings().*

createButtons() – metoda odpowiedzialna za utworzenie przycisków w GUI pozwalające na załadowanie prezentacji, wczytanie ilości slajdów, notatek oraz sterowanie slajdami w prezentacji.

createLabels() – metoda odpowiadająca za utworzenie etykiet informujących o statusie prezentacji, ilości slajdów oraz notatek w niej zawartych.

initStartSettings() – metoda inicjująca ustawienia początkowe w podrzędnym menu.

actionPerformed() – metoda monitorująca działania użytkownika w zakresie GUI i wywołująca odpowiednie metody, po wybraniu odpowiedniego przycisku.

close() – metoda odpowiedzialna za zamknięcie okna podrzędnego menu.

open () – metoda ładująca prezentację do aplikacji przy pomocy metod zawartych w klasie *Extractor.java*.

slide() – metoda wczytująca ilość slajdów do aplikacji.

note() – metoda wczytująca ilość notatek do aplikacji.

previous() – metoda odpowiadająca za przejście do poprzedniego slajdu.

next() – metoda odpowiadająca za przejście do kolejnego slajdu.

Klasa HideFrame.java

Klasa ta odpowiada za utworzenie systemtray'a do aplikacji serwerowej, po jej zminimalizowaniu. Jest to dodatkowa funkcjonalność, która pozwala na korzystanie z opcji aplikacji bez potrzeby operowania w menu głównym. Metody wchodzące w skład tej klasy:

HideFrame() – metoda tworzy systemtray'a oraz wywołuje metodę *createPopUpMenu()*.

createPopUpMenu() – metoda odpowiedzialna za utworzenie menu rozwijanego z poziomu pulpitu, klikając prawym przyciskiem myszy na ikonkę systemtray'a. Pozwala na wybór czcionki, włączenie timer'a lub zresetowanie go.

actionPerformed() – metoda monitorująca działania użytkownika w zakresie GUI i wywołująca odpowiednie metody, po wybraniu odpowiedniego przycisku.

Klasa Extractor.java

Klasa odpowiedzialna jest za wydobywanie danych z prezentacji. Przy jej pomocy ładowana jest prezentacja do aplikacji serwerowej, wczytywane są notatki oraz ilość slajdów. Klasa została napisana w oparciu o bibliotekę *apachepoi*, która umożliwia wydobywanie danych z plików .pptx. Poniżej znajduje się opis poszczególnych metod w klasie:

openPresentation() – otwieranie prezentacji z pliku.

getSlides() – metoda zwracająca liczbę slajdów w prezentacji.

getNotes() – metoda wczytująca notatki i przypisująca je do poszczególnych slajdów.

Klasa LocalHost.java

Klasa ta jest odpowiedzialna za pobranie informacji z komputera użytkownika. Ma na celu zidentyfikowanie jego nazwy oraz adresu IP. Do tego celu korzysta z biblioteki *Inet4Address*.

LocalHost() – metoda przypisująca zmiennym *HOST_NAME* i *HOST_IP* ich wartości wywołując poniższe metody.

Klasa DiscoveryThread.java

W klasie tej utworzono wątek odpowiedzialny za połączenie urządzenia Google Glass z serwerem, tak aby mogło ono pracować niezależnie od reszty aplikacji.

run() - Najpierw tworzony jest socket pod broadcast'owe połączenie. Serwer wyczekuje na zapytanie od aplikacji klienckiej. Po otrzymaniu pakietów, sprawdza je, a następnie wysyła odpowiedź do Google Glass. Ostatecznie tworzony jest socket przeznaczony dla klienta. Wewnątrz tej metody tworzony jest również listener, który odbiera komendy od okularów dotyczące rozpoczęcie prezentacji, przesunięcia slajdu w przód lub w tył oraz zakończenia prezentacji.

getFrame() – metoda odpowiedzialna za przekazanie okna aplikacji, by utworzyć możliwość operacji na nim w klasie *DiscoveryThread.java*

sendText() – metoda odpowiedzialna za przesłanie treści notatek do okularów.

sendFont() – metoda odpowiedzialna za przesłanie czcionki do okularów.

sendChrono() – metoda odpowiedzialna za przesłanie informacji o tym, czy timer powinien być włączony do okularów.

stopListening() – metoda kończy oczekiwanie na zapytanie broadcast'owe.

Aplikacja kliencka

Klasa MainActivity.java

W klasie MainActivity uruchamiane jest główne okno aplikacji, które pozwala wystartować lub zatrzymać usługę. Jest to klasa typu *Activity*, dlatego nadpisane zostały w niej podstawowe metody pozwalające na stworzenie ekranu w aplikacjach na urządzenia z systemem operacyjnym Android.

onCreate() – wywołanie funkcji onCreate() klasy nadrzędnej oraz ustawienie flagi niegaśnięcia ekranu

onAttachedToWindow()

onDetachedFromWindow()

openOptionsMenu()

onCreateOptionsMenu()

onOptionsMenuClosed()

Standardowe funkcje klasy aktywności. Rozszerzana była tylko metoda onOptionsItemSelected() w której zaimplementowano wywołanie odpowiedniej akcji po wybraniu opcji z menu głównego. Uruchomienie usługi spowoduje przejście do aktywności SocketClientActivity natomiast wybór opcji STOP prowadzi do zamknięcia aplikacji.

Klasa SocketClientActivity.java

Klasa SocketClientActivity jest aktywnością, która zarządza całym przepływem aplikacji. To znaczy, że w klasie tej realizowane są po kolei wszystkie kroki potrzebne do odebrania notatek z prezentacji w programie Microsoft PowerPoint i wyświetlenia ich na okularach, a także do sterowania prezentacją.

handleMessage() – funkcja obiektu typu Handler(), która definiuje zachowanie aplikacji na kolejnych etapach dostarczania usługi. Zdefiniowano w niej między innymi wydarzenia:

START_SERVER_DISCOVERY – rozpoczęcie wątku DiscoveryThread w celu wykrycia serwera i ustalenia jego parametrów (adres IP, nr portu) do nawiązania połączenia TCP

SERVER_IP_RECEIVED – otrzymanie parametrów od serwera,

START_TCP_CONNECTION – po udanej próbie odnalezienia serwera, nawiąż z nim połączenie TCP,

TCP_CONN_LOST – utracone połączenie z serwerem,

OPEN_PRESENTATION – otwarcie prezentacji,

START_PRESENTATION – pobranie notatek i wyświetlenie prezentacji na laptopie,

PRINT_NOTES – wyświetlenie części notatek do pierwszego slajdu.

onCreate() – inicjalizacja obiektów, załadowanie ekranu aktywności, uruchomienie wątku łączenia z serwerem, utworzenie obiektu GestureDetector.

GestureDetector.serBaseListener() – implementacja wykrywania gestów. Wykorzystywane zarówno do wyboru opcji z menu, jak i sterowania prezentacją.

LONG_PRESS – wyjście z aplikacji

TAP – otwarcie prezentacji

SWIPE_RIGHT, SWIPE_LEFT – przełączanie pomiędzy slajdami

SWIPE_DOWN – zamknięcie połączenia, zatrzymanie prezentacji

Ponadto klasa SocketClientActivity zawiera prywatną klasę ListeningThread, która implementuje interfejs Runnable. Główna funkcja tej klasy run() służy do asynchronicznego odebrania danych, które przesyła serwer i przygotowania ich do wyświetlenia (zmiana czcionki). To właśnie w tym miejscu uruchamiany jest chronometr.

Klasa DiscoveryThread.java

Zaimplementowane wykrywanie serwera w sieci. Po rozesłaniu pakietów DISCOVER_REQUEST aplikacja przechodzi w stanu nasłuchiwania, aż do momentu odebrania odpowiedzi od serwera DISCOVER_RESPONSE.

Klasa TCPConnectionThread.java

Implementacja połączeniowej komunikacji TCP klient-serwer. Po utworzeniu gniazda sieciowego po obu stronach (po stronie aplikacji klienckiej i serwerowej) możliwe jest bezpośrednie przesłanie danych.

Klasa GetFonts.java

Klasa umożliwia zmianę czcionki wyświetlanych notatek.
GetFonts() – tworzy obiekty typu Typeface na podstawie zasobów znajdujących się w katalogu Assets.

Badania na grupie użytkowników.

W celu sprawdzenia wiarygodnej oceny użyteczności aplikacji w praktyce, przeprowadziliśmy badania na grupie 10 użytkowników. Polegały one na wykorzystaniu rozwiązania poprzez wykorzystanie go przez badanych przy użyciu własnych prezentacji z notatkami. Po zapoznaniu się i przetestowaniu projektu, każdy z uczestników badań musiał wypełnić poniższą ankietę:

Ankieta badawcza – Google Glass jako inteligentny prompter

Witamy! W celu rozwijania naszego projektu przeprowadzamy badania użyteczności i komfortu korzystania ze stworzonej przez nas aplikacji, wspierającej prowadzenie prezentacji. Prosimy o poświęcenie kilku minut na podzielenie się opinią i ewentualnymi uwagami dotyczącymi użytkowania naszego rozwiązania. Odpowiedzi na podane pytania należy oznaczać znakiem „X” w odpowiednim polu.

1. Płeć

   [ ] Kobieta
   [ ] Mężczyzna

2. Jak często zdarza Ci się prowadzić prezentacje?

[ ] Kilka razy w tygodniu
[ ] Raz na tydzień
[ ] Raz na dwa tygodnie
[ ] Raz na miesiąc
[ ] Rzadziej
[ ] Wcale

Pytania dotyczące aplikacji klienckiej (obsługiwanej z poziomu Google Glass)

Prosimy o ocenę poniższych rozwiązań w skali 1-5, gdzie 1 jest wartością najniższą, a 5 najwyższą.

Pytanie	1	2	3	4	5
3. Sterowanie prezentacją.					
4. Widoczność i umiejscowienie notatek wyświetlanych na ekranie Google Glass.					
5. Timer pokazujący czas prezentacji.					
6. Umiejscowienie timer'a w interfejsie.					
7. Przejrzystość interfejsu jako całości.					
8. Szybkość działania aplikacji.					

9. Czy napotkałeś trudności w trakcie połączenia z serwerem?

[ ] Tak
[ ] Nie
Jeśli tak, to jakie?
……

10. Czy napotkałeś komplikacje w trakcie prezentacji?

[ ] Tak
[ ] Nie
Jeśli tak, to jakie?
……

11. Dodatkowe uwagi (w jaki sposób moglibyśmy ulepszyć aplikację, sugestie dodatkowych rozwiązań, napotkane problemy nie uwzględnione w ankiecie).
……
……
……
……
……

Pytania dotyczące aplikacji serwerowej (obsługiwanej z poziomu komputera).

12. Jak oceniasz wygodę obsługi aplikacji serwerowej?

    [ ] Intuicyjna
    [ ] Nieintuicyjna

13. Czy napotkałeś problemy przy wystartowaniu aplikacji serwerowej?

    [ ] Tak
    [ ] Nie
    Jeśli tak, to jakie?
    ……

14. Czy wśród dostępnych rodzajów i rozmiarów czcionek odnalazłeś odpowiadającą dla Ciebie?
    [ ] Tak
    [ ] Nie
    Jeśli nie, to jakiej czcionki/rozmiaru Ci brakuje?
    ……

Pytania dotyczące ogółu projektu.

15. Czy napotkałeś problemy w komunikacji pomiędzy poszczególnymi aplikacjami?

    [ ] Tak
    [ ] Nie
    Jeśli tak, to jakie?
    .....

16. Czy skorzystałbyś z przedstawionego rozwiązania wspomagającego prezentacje, gdyby było dostępne na rynku?

    [ ] Tak
    [ ] Nie
    Jeśli nie, to dlaczego?
    .....

17. Czy spotkałeś się kiedyś z podobnymi rozwiązaniami wspomagającymi prezentację?

    [ ] Tak
    [ ] Nie

18. Dodatkowe uwagi dotyczące ogółu projektu.

.....
.....
.....
.....
.....

19. Czy napotkałeś jakieś trudności w wypełnieniu ankiety?

[ ] Tak
[ ] Nie
Jeśli tak, to jakie?

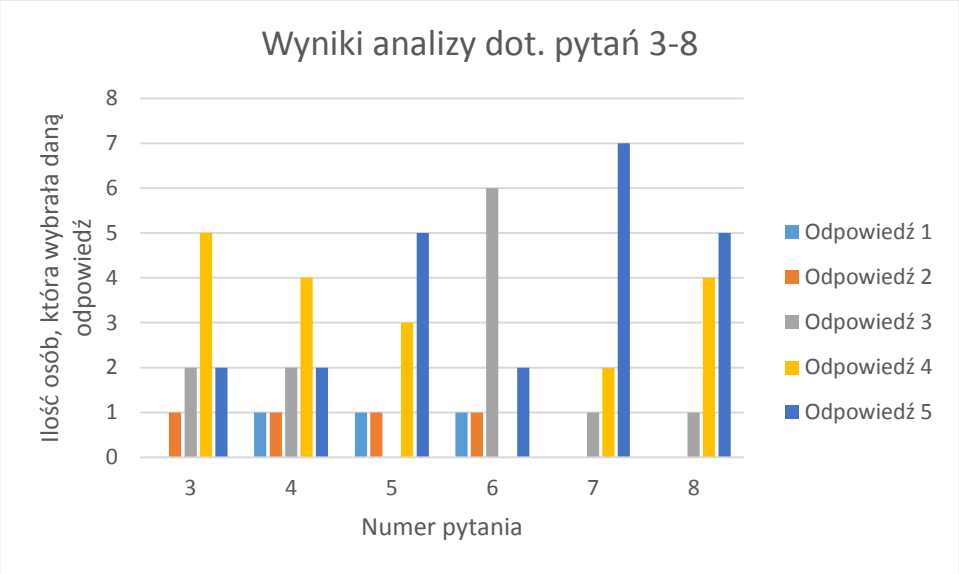
.....
.....
.....

Dziękujemy za uwagę!

Ankietowanymi byli w większości studenci z racji najłatwiejszego dostępu do tej grupy osób. Wybraliśmy po równo mężczyzn i kobiet w celach badawczych. Poniżej przedstawiono wyniki analizy badań.

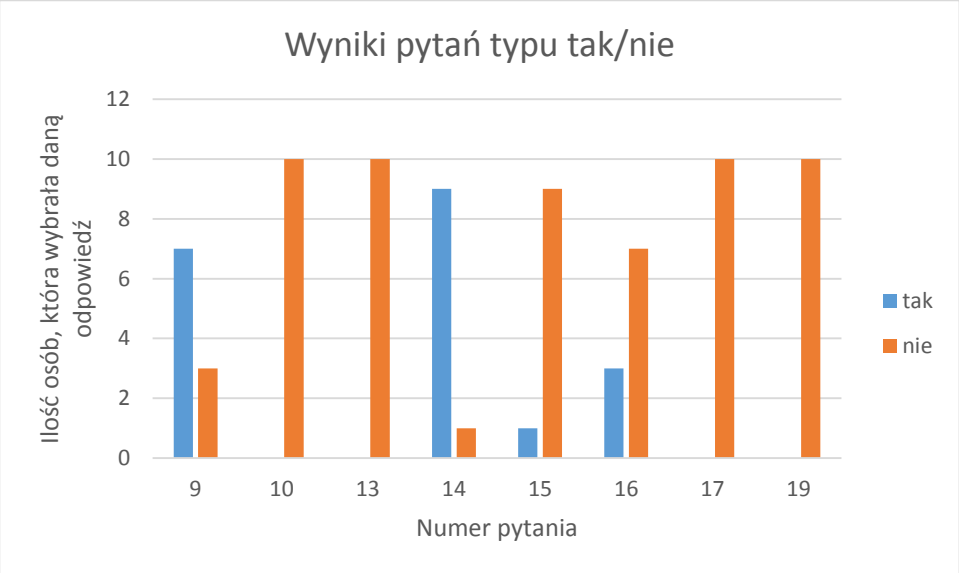
Rys. 7.1 Wykres częstotliwości prowadzenia prezentacji przez badanych.

Uczestnicy badań prowadzili prezentacje z różną częstotliwością w zależności od specyfiki kierunku studiów i zajęć niezwiązanych z kierunkiem.



Rys. 7.2 Wykres opinii badanych na temat aplikacji klienckiej.

Z przeprowadzonej analizy wynika, że większość badanych twierdzi, iż sterowanie prezentacją z poziomu aplikacji klienckiej jest na dobrym poziomie, widoczność i umiejscowienie notatek w okularach również jest na zadowalającym poziomie, timer pokazujący czas prezentacji to znakomite rozwiązanie, umiejscowienie timera wywołuje neutralne odczucia, a przejrzystość interfejsu oraz szybkość działania aplikacji uznana została jako dobra lub znakomita.



Rys. 7.3 Wykres przedstawiający wyniki odpowiedzi na pytania typu tak/nie.

W trakcie nawiązywania połączenia z serwerem u większości ankietowanych zdarzyło się, że nie mógł on zostać odnaleziony, co wymagało ponownego wystartowania serwera z poziomu aplikacji serwerowej. W trakcie prezentacji oraz przy startowaniu serwera z aplikacji nikt nie natknął się na żadne błędy. Prawie wszystkie osoby znalazły odpowiadającą dla siebie czcionkę notatek wyświetlanych na ekranie, co świadczy o sukcesie badań przeprowadzonych w pierwszym semestrze. Komunikacja pomiędzy aplikacjami składowymi odbywała się w większości przypadków bez problemu, w jednym przypadku Google Glass nie zareagował na przesunięcie palcem po panelu. Większość z ankietowanych nie skorzystałaby z tego rozwiązania, gdyby było dostępne na rynku, jednak wszyscy stwierdzili, że to z powodu niewygody w użytkowaniu samego urządzenia. Żaden z ankietowanych nie spotkał się z podobnym rozwiązaniem, jakie oferuje nasz projekt, wcześniej. Na pytanie dotyczące intuicyjności obsługi aplikacji serwerowej wszyscy odpowiedzieli pozytywnie.

Podsumowanie projektu.

Powstałe aplikacje w ramach projektu *Google Glass jako inteligentny prompter* spełniły wszystkie założenia początkowe. W dodatku zostały wprowadzone dodatkowe funkcjonalności takie jak wybór czcionki wyświetlanych notatek na okularach oraz timer informujący prezentera o czasie prowadzonej prezentacji. Głównymi wyzwaniami w trakcie projektu było skomunikowanie urządzenia mobilnego z komputerem, wydobywanie danych z prezentacji w postaci notatek oraz sterowanie samą prezentacją z poziomu Google Glass. Efekt finalny projektu okazał się przydatnym rozwiązaniem usprawniającym prezentacje oraz likwidujący konieczność posiadania notatek w formie papierowej. Największy mankament rozwiązania jest niezależny od członków grupy projektowej, gdyż według większości badanych jest nim samo urządzenie Google Glass. Większość ankietowanych twierdzi, że urządzenie jest niewygodne w obsłudze oraz zbyt drogie, by nabyć je w celu wykorzystania stworzonej w ramach projektu aplikacji. Co do samej aplikacji, większość ankietowanych przyjęła ją pozytywnie, co pozwala sądzić, że projekt zakończony został sukcesem. Niestety nie udało się skompatybilizować rozwiązania tak, by działało z rozszerzeniami .ppt oraz .keynote. Co może być celem dalszego rozwoju projektu. Jedynym błędem, jaki napotkaliśmy w trakcie przeprowadzania badań, sporadycznie występujący brak połączenia z aplikacją serwerową, które zostało odzyskiwane i działało poprawnie w trakcie drugiej próby.

Podsumowanie współpracy z firmą Kainos

Projekt odbywał się we współpracy z firmą *Kainos*. Kooperacja przebiegała pomyślnie, zostały jasno postawione oczekiwania odnośnie aplikacji, na każdym ze spotkań przedstawiciele firmy wspomagali nas swoją wiedzą i doświadczeniem. Dzięki temu byliśmy w stanie określić, czy nasze prace przebiegają we właściwym kierunku. Z racji obowiązków zarówno ze strony przedstawicieli *Kainos*'u jak i członków grupy występowały problemy z ustaleniem i dotrzymaniem terminów spotkań oraz czasami z komunikacją pomiędzy stronami.

Podsumowanie współpracy w grupie

W trakcie okresu prac nad projektem korzystaliśmy z różnych kanałów komunikacyjnych. Firma *Kainos* zaproponowała korzystanie z platformy *Trello*, która nie zdała egzaminu w przypadku grupy czterech znajomych. Uważamy, że w przypadkiej tak małej grupy ludzi, którzy znają się równiez prywatnie, korzystanie z tego rozwiązania jest niepotrzebne. Komunikacja w naszym zespole odbywała się poprzez prywatną grupę na portalu społecznościowym *facebook*, w tym samym serwisie oraz telefonicznie jak również przez spotkania. Drugą zaproponowaną platformą przez firmę był *GitHub*, który z kolei okazał się nieocenioną pomocą przy projekcie. Pozwalał uniknąć wdrożenia błędnych kodów do całości projektu oraz co najważniejsze, zapewniał dostęp do najaktualniejszej wersji aplikacji z każdego komputera, które posiadało połączenie z internetem.