

**Akademia Górniczo-Hutnicza  
im. Stanisława Staszica w Krakowie**

---

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

KATEDRA AUTOMATYKI



**AGH**

**PRACA MAGISTERSKA**

**LESZEK PIĄTEK**

**SPOŁECZNOŚCIOWY, INTERNETOWY SYSTEM  
MONITORINGU**

PROMOTOR:  
dr inż. Sebastian Ernst

Kraków 2012

## **OŚWIADCZENIE AUTORA PRACY**

OŚWIADCZAM, ŚWIADOMY ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ WYKONAŁEM OSOBIŚCIE I SAMODZIELNIE, I NIE KORZYSTAŁEM ZE ŹRÓDEŁ INNYCH NIŻ WYMNIENIONE W PRACY.

.....

PODPIS

**AGH**  
**University of Science and Technology in Krakow**

---

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

DEPARTMENT OF AUTOMATICS



**AGH**

**MASTER OF SCIENCE THESIS**

**LESZEK PIĄTEK**

**SOCIAL, INTERNET MONITORING SYSTEM**

**SUPERVISOR:**  
**Sebastian Ernst PhD**

**Krakow 2012**

Serdecznie dziękuję promotorowi  
dr inż. Sebastianowi Ernstowi za dy-  
stans, wyrozumiałość i cierpliwość.<sup>a</sup>

---

<sup>a</sup>Rozwinięcie podziękowań znajduje się w dodatku  
B Podziękowania autora.

## Spis treści

<b>1. Wprowadzenie.....</b>	<b>7</b>
1.1. Cele pracy .....	8
1.2. Zawartość pracy.....	8
1.3. Wprowadzenie do zwinnej metodyki rozwoju oprogramowania <i>Extreme Programming (XP)</i> .....	9
1.3.1. Założenia.....	9
1.3.2. Zalety .....	10
1.3.3. Wady .....	11
1.3.4. Kiedy nie stosować metodyki .....	11
1.3.5. Cykl życia idealnego projektu .....	12
1.3.6. „Zwinna” specyfikacja wymagań .....	12
1.3.7. Zwinna analiza wymagań .....	13
<b>2. Projekt, etap I – Badanie (Exploration).....</b>	<b>15</b>
2.1. Specyfikacja wymagań .....	15
2.1.1. Specyfikacja głównego zadania systemu .....	15
2.1.2. Karty wymagań dla „pierwszego wydania” .....	17
2.2. Analiza wymagań i dobór rozwiązań .....	29
2.2.1. Drzewo użyteczności (ATAM Usability Tree).....	29
2.2.2. Wstępna architektura systemu .....	30
2.2.3. Technologia.....	33
<b>3. Implementacja prototypu systemu .....</b>	<b>43</b>
3.1. Co zostało zaimplementowane .....	43
3.1.1. Interfejs WWW.....	44
3.1.2. Mobile Broadcaster.....	48
3.2. Co wymaga usprawnienia.....	49
3.3. Dokumentacja.....	51
3.3.1. Uruchomienie systemu.....	51
3.3.2. Testy automatyczne.....	51
3.3.3. Przeprowadzone testy .....	52
3.3.4. Bezpieczeństwo.....	52
3.4. Wydajność P2P Multicast’ing w Adobe Flash Platform .....	53
3.5. Wnioski.....	55
<b>A. L<sup>A</sup>T<sub>E</sub>X, środowisko userstory .....</b>	<b>56</b>

---

A.1. Sposób użycia .....	56
A.2. Przykład użycia .....	56
A.3. Kod środowiska .....	58
A.4. Szablon „Główne zadanie systemu” .....	59
<b>B. Podziękowania autora.....</b>	<b>60</b>

# 1. Wprowadzenie

W ostatnich latach można zauważyć gwałtowny rozwój infrastruktury i technologii sieciowych. Coraz więcej gospodarstw domowych dołącza się do „chmury informacyjnej” jaką jest Internet. Komputery bez dostępu do sieci powoli stają się po prostu bezużyteczne. Internet dzięki Wi-Fi oraz technologiom telefonii komórkowej takich jak HSPA lub jej następcy LTE staje się medium bezprzewodowym, dostępnym wszędzie tam gdzie istnieje zasięg sieci komórkowej.

Wszechobecność i dostępność „wielkiej pajęczyny” powoduje, że coraz częściej korzysta się z niej nie tylko za pomocą komputerów czy laptopów. Nowe „gadżety” takie jak tablety, smartphone’y czy netbook’i stają się nowym interfejsem pomiędzy siecią, a użytkownikiem. Szacuje się, że do roku 2015 większość użytkowników Internetu będzie korzystała właśnie z takich urządzeń mobilnych. Wtedy również, 40% światowej populacji (2,7 miliarda osób) będzie już korzystała z dobrodziejstw dostępu do „natychmiastowej informacji”.<sup>1</sup>

Zwiększenie szybkości dostępu do Internetu umożliwiło stworzenie technologii pozwalających na transmisję strumieniową danych audio/video. Takie serwisy jak YouTube, Vimeo czy MetaCafe już od 2005 roku umożliwiają użytkownikom wysyłanie i udostępnianie swoich nagrań online. Niestety żaden z tych serwisów nie umożliwiał udostępniania strumienia danych wideo w czasie rzeczywistym. Dopiero w 2007 roku powstał Bambuser oraz Qik. W tym roku, dosłownie parę miesięcy temu firma Google połączyła usługę YouTube z Google+ Hangouts umożliwiając równoczesne nadawanie „na żywo”. Wszystkie serwisy te jak i technologie, które wdrożyły, pozwalają dzielić się strumieniem audio/video – wykorzystując do tego jedynie telefon komórkowy lub kamerę podłączoną do komputera.

Naturalną kolejną rzeczą wydaje się wykorzystanie Internetu celem stworzenia gigantycznego systemu monitoringu. Można sobie wyobrazić system do którego podłączono istniejącą infrastrukturę kamer CCTV i udostępniono ich strumień video online, tak aby każdy obywatel mógł monitorować to co się dzieje w jego okolicy. System można by było rozszerzyć o kamery podłączane przez użytkowników lub ich najnowocześniejsze „gadżety” czyli tablety czy smartphone’y. Tak właśnie autor niniejszej pracy wyobraża sobie „Społecznościowy, internetowy system monitoringu”.

---

<sup>1</sup>Dane w oparciu o artykuł [Kim11]

## 1.1. Cele pracy

Głównym celem pracy, a co za tym idzie i najbardziej czasochłonnym celem pracy jest implementacja ograniczonego prototypu systemu określanego mianem „Społecznościowy, internetowy system monitoringu”.

Dodatkowym celem, bo koniecznym do realizacji głównego, jest stworzenie specyfikacji oraz analizy wymagań systemu określonego jako „Społecznościowy, internetowy system monitoringu”. Na każdym etapie przygotowywania projektu, chciano wykorzystać jak najwięcej metodyk i technologii „zwinnych”. Dlatego też bazując na pracy starszych kolegów ([JM06]), jak i doświadczeniu w zwinnym specyfikowaniu systemu doktorantów ([MK09]), podpierając się literaturą fachową ([Bec99]) autor chce zaproponować własną „zwinną specyfikację i analizę wymagań”. W szczególności pokazać jej zalety oraz wady względem podejścia konwencjonalnego. Uwytklić przypadki, w których metodyki zwinne nie są dobrym rozwiązaniem.

## 1.2. Zawartość pracy

W pierwszym rozdziale pracy opisano teorię związaną ze „zwinną” metodyką tworzenia oprogramowania. Umieszczono jej wady, zalety oraz wyszczególniono kiedy nie należy stosować metodyk zwinnych. Tutaj znalazło się również miejsce na opisanie idealnego cyklu życia idealnego „zwinnego” projektu oraz autorską propozycję przeprowadzania akwizycji i analizy wymagań takiego przedsięwzięcia.

Kolejny rozdział pracy poświęcony jest akwizycji i analizie wymagań tytułowego „Społecznościowego systemu monitoringu”. W tej części pracy znajdują się więc takie elementy jak karty wymagań, ekrany, wstępna architektura systemu czy opis technologii proponowanych do zastosowania. Przy realizacji ww. elementów wykorzystano zaproponowaną i określoną w rozdziale pierwszym metodykę.

Ostatni z rozdziałów poświęcony jest najbardziej pracochłonnemu elementowi pracy – implementacji ograniczonego prototypu systemu. Można znaleźć tutaj informacje szczegółowe na temat tego co udało się zaimplementować, z czym były problemy, jak sprawdziły się technologie i architektura wymieniona w rozdziale wcześniejszym.



## 1.3. Wprowadzenie do zwinnej metodyki rozwoju oprogramowania *Extreme Programming (XP)*<sup>2</sup>

„Wytwarzając oprogramowanie i pomagając innym w tym zakresie,  
odkrywamy lepsze sposoby wykonywania tej pracy.

W wyniku tych doświadczeń przedkładamy:

**Ludzi i interakcje** ponad procesy i narzędzia.

**Działające oprogramowanie** ponad obszerną dokumentację.

**Współpracę z klientem** ponad formalne ustalenia.

**Reagowanie na zmiany** ponad podążanie za planem.

Doceniamy to, co wymieniono po prawej stronie,  
jednak bardziej cenimy to, co po lewej.”

— Kent Beck et al., <http://agilemanifesto.org>

Przy tworzeniu projektu wchodzącego w skład niniejszej poczyniono kilka założeń. Większość z nich dotyczy sposobu prowadzenia projektu – dokumentacji, testów, komunikacji w grupie. W następnym punkcie znajduje się lista tych założeń.

Projekt nie spełniający ich, nie będzie mógł być prowadzony „zwinnie”. Listę można traktować jako swojego rodzaju zbiór dobrych praktyk metodyki rozwoju oprogramowania zwanej *Extreme Programming* lub w skrócie *XP* – Programowanie Ekstremalne. Oczywiście ze względu na samodzielny charakter pracy, autor nie mógł sprawdzić wszystkich założeń w praktyce.

### 1.3.1. Założenia

Niżej przedstawiono podstawowe założenia metodyki *Extreme Programming* bez których metodyka nie tyle nie będzie działała, co nie będzie można powiedzieć, że w projekcie zastosowana jest metodyka *XP*.

**Klient jest zawsze dostępny** Osoba która wie jak system ma działać od strony użytkownika jest zawsze dostępna dla programistów, forma komunikacji nie jest narzucona, natomiast preferowana jest twarzą w twarz. Osoba ta nie jest potrzebna tylko na początku projektu lecz *przez cały okres* jego tworzenia i rozwoju.

**Klient ma zawsze możliwość zmiany** Nie ma rzeczy której nie da się zmienić w systemie, klient może w każdym momencie zmienić wymagania systemu. Grupa projektowa musi reagować na te zmiany.

**Projekt jest prowadzony w krótkich iteracjach** Jedno do trzy-tygodniowe okresy czasu, po których klient otrzymuje chciane funkcjonalności, wykonania których podjęliśmy się w zadanym okresie czasu.

**Testowanie automatyczne** Każdy nowy element systemu musi posiadać napisane testy automatyczne (jednostkowe, funkcjonalne) – system można w każdym momencie automatycznie przetestować. Najlepiej testy pisać przed implementacją nowego elementu.

**Projektujemy, planujemy zawsze proste minimum** Nie powinno planować się na wyrost. Należy określać minimum, które będzie spełniało wymagania klienta w danej iteracji. Zawsze powinno trzymać się jak najmniej złożoność systemu, nawet kosztem dodatkowych zmian.

<sup>2</sup>Głównym źródłem informacji na którym ten rozdział jest książka [Bec99]. Autor tej książki uważany jest za twórcę metodyki *XP*.

**Ciągła integracja** Poprawki nanoszone są cały czas na istniejący, system. Integracja następuje wręcz kilka razy dziennie. Wymagany jest reżim testowy – integrowany jest tylko ten kod, który ma napisane testy automatyczne oraz nie blokuje wykonania żadnego z dotychczasowo napisanych testów.

**Brak specjalizacji** Żadna osoba z grupy projektowej nie powinna specjalizować się w jakiejś funkcji (programista, architekt, integrator, analityk). Wszyscy biorą czynny udział we wszystkich aspektach projektu.

**Grupa prowadząca projekt nie jest zbyt liczna** Grupa w której tworzony jest projekt lub pod-projekt nie może być zbyt liczna, ułatwia to komunikację. Każdy większy projekt da się podzielić na szereg mniejszych pod-projektów. Nie ma określonego limitu górnego, jeżeli występują problemy w komunikacji najczęściej grupa projektowa jest zbyt liczna.

**Komunikacja twarzą w twarz** Preferowaną formą komunikacji w grupie projektowej jest komunikacja twarzą w twarz. Tylko przy takiej rozmowie uczestnicy projektu nie są rozpraszeni przez inne rzeczy i mogą skupić się na zadanym temacie. Taki sposób komunikacji ułatwia znajdowanie błędów i niejasności we wczesnej fazie projektu.

**Wspólne programowanie, projektowanie** Faworyzuje się tworzenie wszystkich elementów systemu w parach. Zapewniając w ten sposób ciągłe badanie jakości kodu, czy też poprawności architektury systemu.

**Architektura jest zmienna** Architektura jest czymś co się zmienia wraz z rozwojem projektu i funkcjonalności wymaganej przez klienta. Może się zmienić na każdym etapie projektu.

**Kod i testy są dokumentacją** Nie ma prowadzonej dodatkowej dokumentacji implementacyjnej np. dla programistów. Dobrze napisane testy, kod z komentarzami oraz ostatecznie rozmowa z innymi uczestnikami projektu jest najlepszą dokumentacją aktualnego stanu systemu.

### 1.3.2. Zalety

Metodyka *Extreme Programming* zmienia całkowicie podejście do sposobu tworzenia oprogramowania. Głównie przez umieszczenie klienta, w centrum prowadzonego projektu i jego ciągłe zaangażowanie na całym etapie tworzenia i rozwoju systemu. To on może w dowolnym momencie wprowadzić praktycznie dowolne zmiany. Zyski jakie daje metodyka zwinna *XP* można rozpatrywać na wielu płaszczyznach:

**Koszty** Metodyka daje nowe możliwości związane z liczeniem kosztów (per iteracja, per funkcjonalność). Przez krótkie iteracje i działające oprogramowanie łatwiej rozliczać się z klientem, a sam klient chętniej płaci. Długoterminowo unikamy kosztów wynikających z błędów początkowego projektowania, czy też zmian w projekcie, których nie jesteśmy w stanie uniknąć.

**Czas** Zarządzanie czasem w krótkich terminach iteracyjnych jest dużo bardziej wydajne i bardziej przewidywalne niż planowanie całego projektu od początku do końca. Brak konkretnej daty zakończenia projektu umożliwia uniknięcie „marszów śmierci”, czy przesuwania terminów. Klient sam zarządza tym co ma być zrobione i jakim kosztem czasowym jest to obciążone. Reżim testowy i ciągła integracja powoduje, że oprogramowanie dostarczane jest szybko, i co bardzo ważne dla klienta – jest to oprogramowanie spełniające jego aktualne wymagania.

**Jakość** Ciągła integracja i obecność testów powoduje, że pomimo możliwych ciągłych zmian kod w większości przypadków działa, czyli jest wysokiej jakości. Jakość kodu gwarantowana jest również przez programowanie w parach. Zastępuje ono cykliczne sprawdzanie kodu oraz umożliwia uniknięcie problemów związanych z błędną architekturą systemu, które najczęściej pojawiają się bardzo późno.

**Zakres** Bardzo łatwo zmienić zakres systemu w trakcie jego tworzenia, wymaga się jedynie ustalenia w której iteracji ma być on zwiększony lub pomniejszony. Jako, że nie mamy ustalonych terminów całości projektu, zmiana zakresu nie jest dla nas.

**Brak specjalizacji** Ciągła komunikacja werbalna w projekcie zapobiega specjalizacji poszczególnych członków zespołu. Każdy powinien wiedzieć jak najwięcej o systemie. Takie podejście gwarantuje niezależność w przypadku odejścia członka „specjalisty”.

**Porażka projektu** Szybkie dostarczanie działającego, funkcjonalnego oprogramowania ma duży wpływ na zadowolenie klienta. Przekłada się to bezpośrednio na poczucie satysfakcji oraz pozytywną atmosferę wśród członków zespołu. Dzięki temu ludzie są mniej skłonni do zmiany miejsca zatrudnienia ze względu na stres, ciągle przekraczanie terminów czy brak satysfakcji z wykonywanej pracy.

### 1.3.3. Wady

Jak każda metodyka, także i ta „zwinna” posiada swoje wady. Warto znać je przed rozpoczęciem projektu, w którym planuje się ją wykorzystać. Główne z nich są wymienione niżej.

**Ciągła dostępność klienta** Klient lub osoba wyznaczona przez niego, która zna potrzeby systemu musi być dostępna programistom „na zawołanie” i ściśle z nimi współpracować.

**Konsekwencja** Decydując się na „XP” nie możemy zrezygnować z założeń (wspomniane szczegółowo w rozdziale 1.3.1 Założenia). Brak konsekwencji spowoduje w większości przypadków porażkę projektu.

**Brak konkretnej daty zakończenia projektu** Podejście iteracyjno-wymaganiowe powoduje, że znany jest jedynie czas zakończenia aktualnej iteracji w skład której wchodzi konkretne karty wymagań. Ciągła możliwość zmiany zakresu i funkcjonalności systemu powoduje, że nie można określić daty zakończenia projektu.

### 1.3.4. Kiedy nie stosować metodyki

Czasami metodyki *Extreme Programming* nie można zastosować ze względu na charakter prowadzonego projektu lub przyzwyczajenia klienta. Główne wskazówki kiedy *XP* **nie nadaje się** jako metodyka prowadzenia projektu opisane są poniżej.

**Brak dostępności klienta** Jeżeli wiadomo, że klient nie będzie brał czynnego udziału w projekcie. Doprowadzi to do sytuacji, w której tworzone oprogramowanie będzie oprogramowaniem niezgodnym z wymaganiami klienta.

**Duża grupa projektowa** Grupa projektowa *XP* nie powinna być większa niż 10 osób. Więcej osób generuje problemy w komunikacji. Częściowym rozwiązaniem może być podział systemu na podsystemy, przygotowywane przez kilka mniejszych grup projektowych.

**Długie wykonywanie testów** Kompilacja i wykonanie testów automatycznych trwa dłużej niż dzień (24 godziny). Co praktycznie uniemożliwia systematyczną „ciągłą integrację”.

**Brak środowiska testowego innego niż produkcyjny** W sytuacji kiedy koszty środowiska testowego są bardzo wysokie, klient nie będzie chciał wydawać drugi raz dużych pieniędzy. Może się też zdarzyć, że z innych względów nie będziemy w stanie zagwarantować środowiska testowego, co automatycznie wyklucza testy automatyczne i „ciągłą integrację”.

### 1.3.5. Cykl życia idealnego projektu

Każda metodyka ma etapy w jakim może znaleźć się projekt. W tym rozdziale przybliżono idealny cykl życia projektu *XP*.

1. **Badanie (Exploration).** Faza w której pobiera się od klienta główne wymagania systemu, testuje możliwe do wykorzystania technologie, bada się ich wydajność i użyteczność w projekcie. Jeżeli wykorzystywana technologia jest nowa dla zespołu, testuje się jej wykorzystanie na jakimś małym projekcie powiązonym z docelowym systemem.
2. **Planowanie (Planning).** Faza w której grupa projektowa szacuje z klientem terminy głównych wymagań systemu. Klient określa priorytet, a grupa projektowa czas potrzebny na wykonanie poszczególnych kart wymagań.
3. **Iteracje do pierwszego wydania (Iterations to First Release).** Faza w której w iteracjach jedno do trzytygodniowych przygotowuje się system zgodnie z kartami wymagań. Koniec każdej iteracji powinien być „małym świętem” – darmowa pizza, wolny dzień w pracy etc.
4. **Wdrażanie do produkcji (Productionizing).** Faza w której system jest wdrażany do produkcji. Na tym etapie należy pamiętać o sposobie na łatwą integrację zmian – zarówno kodu jak i migracji danych oraz systemie testowym. Pod koniec tego etapu należy zrobić huczną imprezę!
5. **Utrzymanie i konserwacja (Maintenance).** Jest to najdłuższa faza projektu „XP”. Można by właściwie powiedzieć jego naturalny stan. W tej fazie iteracyjne, wdrażane są szczegółowe wymagania klienta, czy też nowe funkcjonalności odpowiadające jego aktualnym potrzebom.
6. **Zakończenie (Death).** Jest to faza w której projekt uznawany jest za zakończony. Może to być sytuacja idealna, w której klient nie ma potrzeby dalszej zmiany oprogramowania – dostarczono mu produkt idealny. Może to być również mniej przyjemna sytuacja – system nie jest w stanie spełnić nowych wymagań stawianych przez klienta. Tak czy inaczej, dobrym pomysłem jest zorganizowanie „stypy” na której trzeba poruszyć problem, tego: „Co można było zrobić lepiej?” i wyciągnąć na tej podstawie wnioski mogące się przydać w nowym projekcie.

### 1.3.6. „Zwinna” specyfikacja wymagań

Do tej pory nie znaleziono idealnego, uniwersalnego rozwiązania na pobieranie wymagań od klienta. W większości przypadków trzeba pewnego czasu, aby wypracować wspólny język z klientem i nauczyć się razem z nim współpracować przy akwizycji wymagań.

Podchodząc zwinnie do projektowania systemu na początku należy pobrać od klienta tylko te wymagania, które są konieczne do osiągnięcia minimalnych celów biznesowych – głównego zadania systemu. Wszystkie sprawy poboczne, należy oczywiście zapisywać, lecz zostawić na później, aż projekt przejdzie do etapu *Utrzymanie i konserwacja* (więcej w rozdziale 1.3.5 Cykl życia idealnego projektu).

Celem zwinnej specyfikacji wymagań jest stworzenie wymagań **minimalnego systemu**, spełniającego **wszystkie podstawowe funkcje biznesowe** zdefiniowane przez klienta, który można **wdrożyć, używać i rozwijać**.

#### Narzędzia

Zwinna specyfikacja wymagań *Extreme Programming* posiada szereg prostych narzędzi, które upraszczają proces jej przygotowania. Są one wraz z opisami wymienione poniżej.

**User Stories** W [MK09] określane jako karty wymagań. Są to krótkie, ogólnikowe historyjki opisujące jakąś funkcjonalność systemu.

**Screens** W [MK09] określane jako ekrany. Są to szkice obrazujące ogólny układ przycisków, formularzy, tabel mogących pojawić się w karcie wymagań

**Acceptance Tests** W [MK09] określane jako testy akceptacyjne/scenariusze. Są to opisy słowne prawidłowego zachowania systemu dotyczące kart wymagań, swojego rodzaju scenariusz karty wymagań na bazie których programiści mogą stworzyć testy (o testach akceptacyjnych w *XP* można więcej przeczytać w [Jef00] pod hasłem *Acceptance tests*)

Dokumenty te nie mają konkretnej formy. Zwinna metodyka pozwala dostosować ją do własnych wymagań, które będą działały najlepiej dla wykorzystujących metodykę programistów oraz ich klienta. Karty wymagań, scenariusze oraz ekrany należy spisywać na małych karteczkach jednego formatu. Przygotowując powyższe „dokumenty” należy pamiętać o poniższych zasadach.

**Prostota** Złożone scenariusze czy karty wymagań należy rozbić na kilka mniejszych, dzielenie musi być zrobione z klientem w formie rozmowy – nigdy przez samego programistę. Karty wymagań powinny być do wykonania w ciągu jednego do dwóch tygodni czasu programisty.

**Klient jest autorem kart wymagań jak i testów akceptacyjnych/scenariuszy** Programista może jedynie pokazać jak należy pisać kartę wymagań, tak żeby wypracować z klientem wspólny język, może pomagać w podzieleniu złożonych wymagań na mniejsze. Dopuszcza się sugerowanie możliwych scenariuszy, natomiast sama reakcja systemu musi być już określona przez klienta

**Ekran powinien mieć swoją sekwencję** Jak istnieje sekwencja ekranów, należy pamiętać o nadaniu numerów odpowiadających kolejności ich występowania.

### Cyfryzacja i zarządzanie

W związku z potrzebą dołączenia kart wymagań, związanych z nimi testów akceptacyjnych oraz ekranów do niniejszej pracy, trzeba było wymyślić sposób na ich cyfryzację. Dodatkową zaletą formy elektronicznej jest fakt, że ułatwia ona wymianę informacji pomiędzy uczestnikami projektu, pomoże uniknąć sytuacji zagubienia lub zniszczenia jakiegoś elementu kart wymagań, ułatwi ich zarządzaniem i przetrzymywaniem np. w wykorzystywanym systemie kontroli wersji.

Wykorzystywanie  $\LaTeX$  u do składu pracy nakierowało na pomysł stworzenia szablonu  $\LaTeX$ , który można by było wykorzystać celem konwersji kart wymagań i ich elementów analogowych do dokumentu cyfrowego. Łatwe oddzielenie warstwy prezentacji od samej treści oraz możliwość eksportu do formatu PDF umożliwiłoby również tworzenie w prosty sposób ładnego dokumentu dla klienta po spotkaniach na których pobierane są od niego wymagania.

Tak powstał tzw. *package* udostępniający środowisko `userstory`. Szczegółowy opis tego środowiska znajduje się w rozdziale A  $\LaTeX$ , środowisko `userstory`.

### 1.3.7. Zwinna analiza wymagań

Zwinna analiza wymagań ma miejsce w fazie *Badanie (Exploration)* projektu (więcej w rozdziale 1.3.5 Cykl życia idealnego projektu), po tym jak ustalono z klientem główne zadanie systemu oraz sporządzono dla niego wszystkie karty wymagań. Cele tej fazy są wymienione poniżej.

**Testowanie technologii** Poprzez budowanie prototypów, testuje się wykonalność oraz wydajność konkretnej technologii. Testy służą wybraniu najbardziej odpowiedniej technologii. Jeżeli uruchomienie jakiejś technologii trwa dłużej niż tydzień, należy zastanowić się nad alternatywami.

**Testowanie architektury** Najczęściej system można zbudować na kilka sposobów, z pomocą może przyjść znowu prototypownie. W grupach 2-3 osobowych tworzone są równoległe 2-3 podejścia do architektury systemu. Porównanie umożliwia wybranie tej najlepszej.

**Poznanie limitów wydajnościowych rozwiązania** Bardzo ważnym czynnikiem, który trzeba wziąć pod uwagę jest wydajność i odnieść ją wymagań stawianych przez klienta.

**Testowanie skalowalności** Równie ważnym czynnikiem jest sposób w jaki skaluje się projektowane rozwiązanie. Zapewnienie stałej, liniowej, bądź logarytmicznej skalowalności jest ideałem.

**Wybór narzędzi umożliwiających ciągłą integrację** Jednym z ważniejszych wymagań *XP* jest zdolność do ciągłej integracji. Należy wybrać system kontroli wersji kodu źródłowego, narzędzie do przeprowadzania testów automatycznych oraz narzędzie umożliwiające łatwą i pewną migrację danych.

**Wybór narzędzi wspomagających prowadzenie projektu** Powinno się również pomyśleć o narzędziach wspomagających zarządzanie projektem, czy też ułatwiających komunikację między uczestnikami projektu.

Zgodnie z założeniami „XP” (więcej w rozdziale 1.3.1 *Założenia*), wszystkie ustalenia sporządzone w tej fazie mogą ulec zmianie na każdym późniejszym etapie prowadzenia projektu. Zwinna analiza wymagań nie zamyka możliwości późniejszej zmiany technologii, architektury czy dowolnego z elementów wspomagających projekt.

## 2. Projekt, etap I – Badanie (Exploration)

Zgodnie z cyklem życia „XP” (więcej w rozdziale 1.3.5 Cykl życia idealnego projektu), projekt zaczyna się od akwizycji ogólnego, głównego zadania systemu. Proponuje się w tym celu wykorzystać przygotowany szablon (więcej w rozdziale A.4 Szablon „Główne zadanie systemu”). Zawiera on nie tylko najważniejszą część, czyli ogólny opis głównego zadania systemu, ale także szereg pomocnych pytań pomagających w realizacji systemu zgodnie z oczekiwaniami klienta. Szablonu nie należy traktować jako idealnego, a raczej jako propozycję lub punkt wyjścia do rozwinięcia własnego dokumentu wykorzystywanego do akwizycji ogólnych wymagań w swoich projektach.

### 2.1. Specyfikacja wymagań

Specyfikacja wymagań jest bardzo ważnym elementem w cyklu tworzenia systemu. Ewentualne niedociągnięcia w tej fazie mogą rzutować na cały projekt. Celem zwinnej specyfikacji wymagań jest stworzenie wymagań minimalnego systemu, spełniającego wszystkie podstawowe funkcje biznesowe zdefiniowane przez klienta, który można wdrożyć, używać i rozwijać – więcej można przeczytać w rozdziale 1.3.6 „Zwinna” specyfikacja wymagań.

#### 2.1.1. Specyfikacja głównego zadania systemu

Sformułowanie głównego zadania systemu pozwala skupić się generowaniu kart wymagań związanych z nim, a nie ze sprawami pobocznymi. Pozwala na uzmysłowienie grupie projektowej jaka jest główna część systemu wokół której należy się skupić. Niżej przedstawiono szablon (więcej w rozdziale A.4 Szablon „Główne zadanie systemu”) wypełniony razem z klientem na jednym z pierwszych spotkań dotyczących wymagań systemu.

##### **Karta wymagań „Główne zadanie systemu”**

Użytkownik za pomocą smartphona z kamerą działającego pod systemem Android lub za pomocą przeglądarki internetowej na komputerze stacjonarnym i kamery podłączonej do niego, ma możliwość udostępnienia on-line aktualnego obrazu i dźwięku.

##### **Pytania do klienta „Główne zadanie systemu”**

- **Kto jest użytkownikiem systemu?** Użytkownikiem systemu może być każdy kto ma przeglądarkę. Aby udostępnić swoją kamerę dodatkowo użytkownik musi być zalogowany, a wcześniej zarejestrowany w systemie. Aby korzystać z reszty funkcji serwisu nie trzeba być zalogowanym.

- **Jakie urządzenia muszą współpracować z systemem?** Wszystkie komputery wyposażone w przeglądarkę IE 7.0+, Firefox 3+, Chrome, Opera 9+. Dodatkowo system ma działać na telefonach komórkowych / tabletach z systemem Android.
- **Czy jest wymóg użycia konkretnej technologii?** Nie jest wymagana żadna konkretna technologia. Wymaganiem jest wykorzystanie technologii Open Source – wszędzie tam gdzie jesteśmy w stanie zastąpić komercyjne rozwiązania.
- **Jaka jest wymagana skalowalność systemu?** System musi być skalowalny, idealnie by było, gdyby obraz z kamery udostępniany przez jednego użytkownika, mógłby być odbierany przez nieskończoną liczbę ludzi. Na pewno w jakiś sposób system musi reagować automatycznie na zwiększony ruch w obrębie jednego stream'a.
- **Czy są jakieś wymagania dotyczące środowiska produkcyjnego w jakim ma działać system?** System docelowo będzie wdrożony na odpowiednim sprzęcie, testy powinien być w stanie odpalić się na średniej klasy laptopie. Im mniejsze wymagania tym lepiej.
- **Czy system ma współpracować z innymi systemami lub udostępniać coś innym systemom?** Duży nacisk będzie kładziony na integrację z serwisami społecznościowymi, głównie chodzi o Facebook'a. Będzie można go wykorzystać celem rejestracji i logowania w serwisie, czy też automatycznego udostępniania stream'u na swojej Facebook'owej ścianie.
- **Czy istnieją systemy podobne do tworzonego?** Jeżeli chodzi o sposób prezentacji interfejsu WWW to najbardziej zbliżony do tego co chce się osiągnąć jest Bambuser. Podobnie ma się sprawa funkcjonalności na telefonie z systemem Android – aplikacja Bambuser.
- **Czy są jakieś inne specjalne wymagania, które nie wynikają z funkcji jakie powinien posiadać system (wymagania нефункционалне)?** Szybki, skalowalny, bezpieczny, wykorzystujący technologie Open Source.



### 2.1.2. Karty wymagań dla „pierwszego wydania”

Akwizycja głównego zadania systemu pozwala na przejście do kolejnego etapu, w którym przygotowuje się wszystkie karty wymagań dla „pierwszego wydania” (więcej w rozdziale 1.3.5 Cykl życia idealnego projektu). Faza ta trwa do momentu kiedy klient nie będzie już miał pomysłu na karty wymagań związane z głównym zadaniem systemu lub grupa projektowa nie będzie w stanie lepiej oszacować czasu potrzebnego na wykonanie danej karty wymagań bez rozpoczęcia jej implementacji.

#### Karta wymagań „Strona główna dla niezalogowanego użytkownika”

Użytkownikowi niezalogowanemu w systemie, po wejściu na główną domenę serwisu, zostaje zaprezentowana zawartość w postaci:



Rysunek 2.1: Strona główna dla niezalogowanego użytkownika

1. odnośnik przekierowujący zawsze na stronę główną serwisu
2. wyszukiwarka umożliwiająca wyszukiwanie udostępnionych streamów audio/video po kategoriach i lokalizacji
3. dwa odnośniki, dzięki którym pokazuje się formularz logowania lub rejestracji
4. mapa pokazująca grupy streamów audio/wideo

**Testy akceptacyjne „Strona główna dla niezalogowanego użytkownika”**

- Użytkownik po wejściu na stronę główną widzi mapę swojej okolicy (fizycznej lokalizacji). Lokalizację możemy oszacować za pomocą IP lub jak jest taka możliwość w każdy inny dokładniejszy sposób.
- Użytkownik może dowolnie przesuwać mapą. Mapa reaguje na przesuwanie, oraz są na niej aktualizowane punkty streamów audio/video.

**Karta wymagań „Strona główna dla zalogowanego użytkownika”**

Użytkownikowi zalogowanemu w systemie, po wejściu na główną domenę serwisu, zostaje zaprezentowana zawartość w postaci:



Rysunek 2.2: Strona główna dla zalogowanego użytkownika

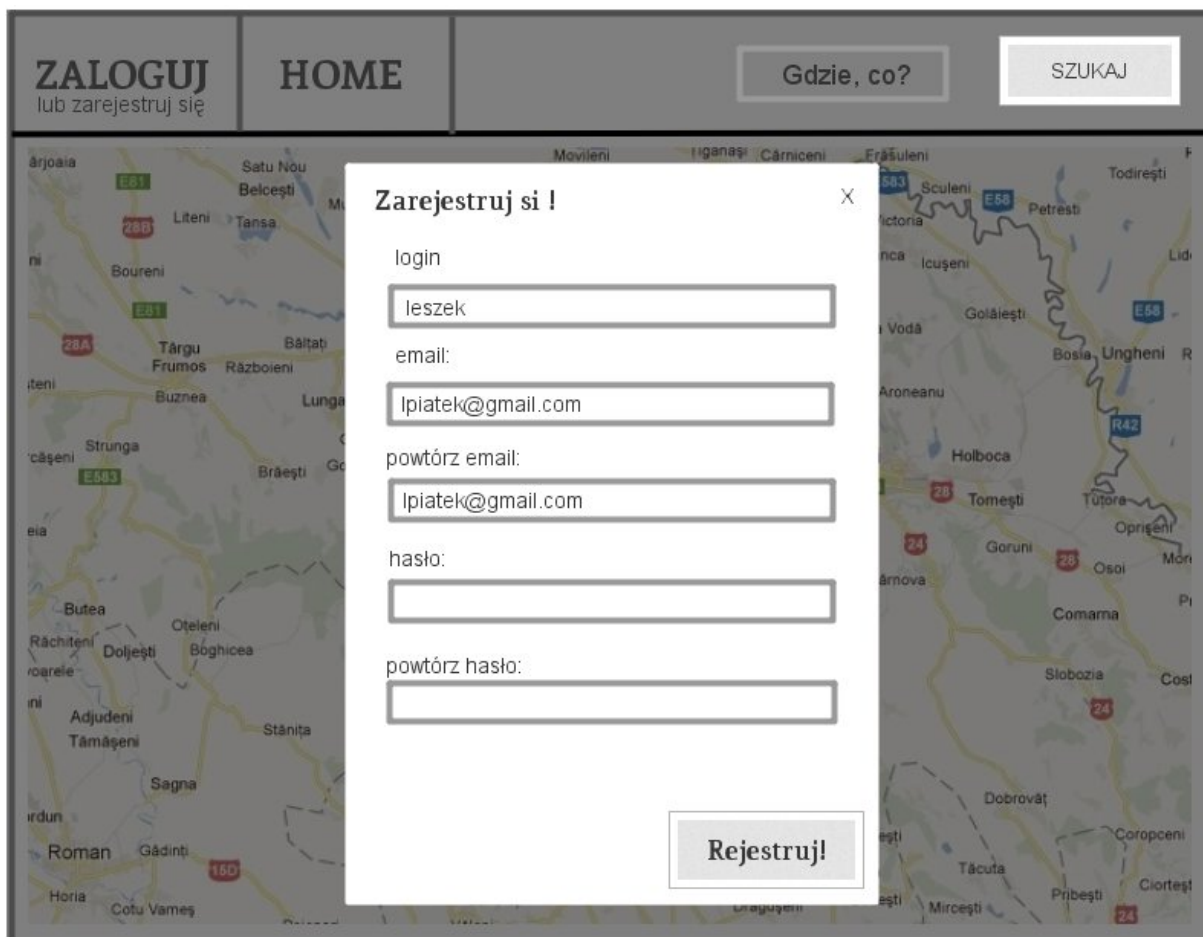
1. avatar pobrany z gravatar.com oraz nazwa zalogowanego użytkownika
2. wpis w menu przenoszący do podstrony ustawień zalogowanego użytkownika
3. wpis w menu przenoszący do podstrony udostępniania streamu audio/video

**Testy akceptacyjne „Strona główna dla zalogowanego użytkownika”**

- Po kliknięciu w menu na pozycję z nazwą zalogowanego użytkownika, pojawiają się poniższe wpisy w menu.

### Karta wymagań „Rejestracja”

Użytkownik po naciśnięciu odnośnika rejestruj, zostaje przekierowany do formularza rejestracji:



The screenshot shows a web interface with a map background. At the top, there are navigation links: 'ZALOGUJ lub zarejestruj się', 'HOME', 'Gdzie, co?', and 'SZUKAJ'. A modal window titled 'Zarejestruj si !' is open in the center. It contains the following fields:

- login:
- email:
- powtórz email:
- hasło:
- powtórz hasło:

A 'Rejestruj!' button is located at the bottom right of the modal window.

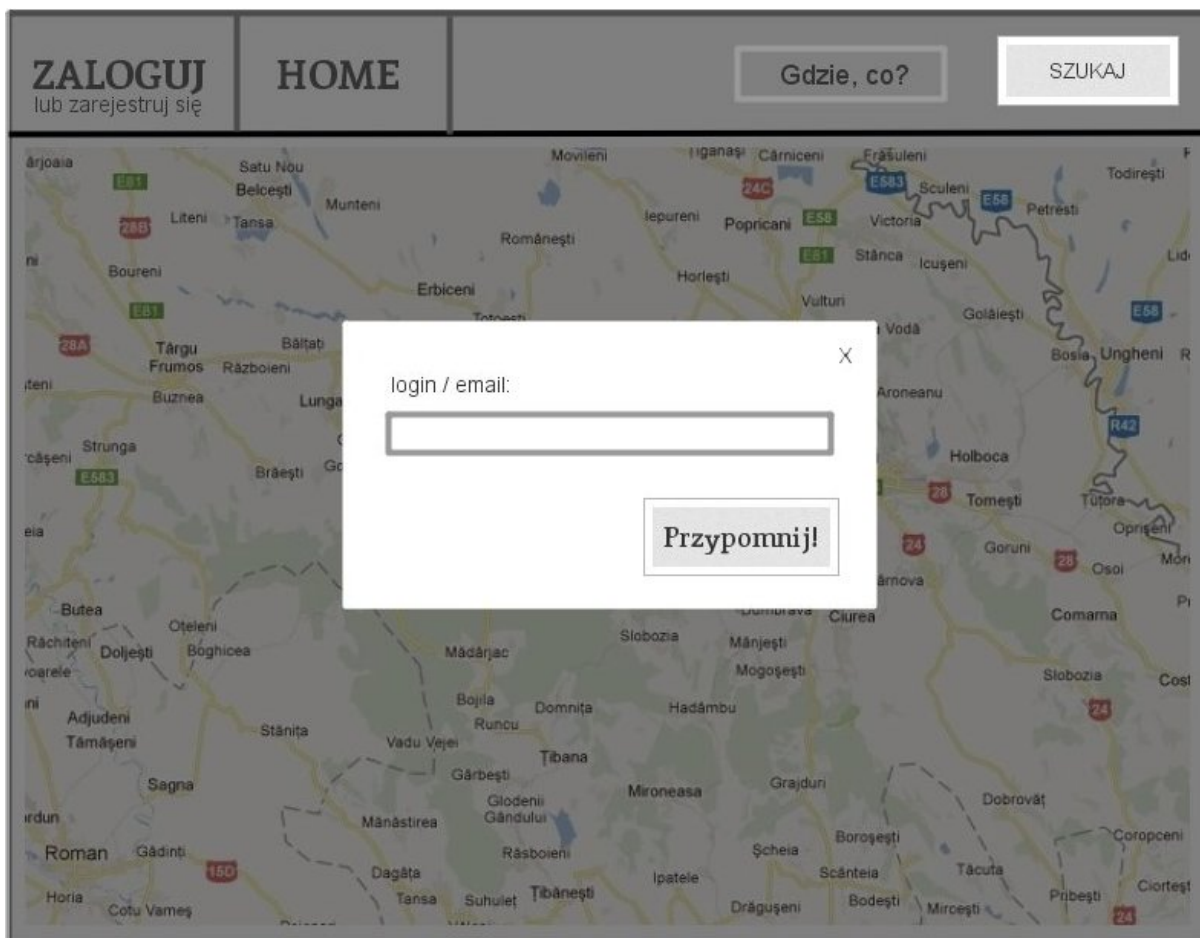
Rysunek 2.3: Formularz rejestracji.

### Testy akceptacyjne „Rejestracja”

- Pole email i powtórz email muszą być takie same, inaczej wyświetlany jest błąd, użytkownik proszony jest o poprawę pól.
- Pole hasło i powtórz hasło muszą być takie same, inaczej wyświetlany jest błąd, użytkownik proszony jest o poprawę pól.
- Pole loginu musi być unikalne w bazie. W przypadku wystąpienia w bazie zwracamy użytkownikowi błąd.
- Pole email musi być unikalne w bazie. W przypadku wystąpienia w bazie zwracamy użytkownikowi błąd.
- Po prawidłowym wypełnieniu formularza, użytkownik dostaje informację o konieczności potwierdzenia swojego adresu e-mail. Aby potwierdzić rejestrację musi kliknąć w link aktywacyjny wysłany na rejestrowane konto pocztowe, w ciągu 7 dni od momentu rejestracji. W innym wypadku konto nie zostanie aktywowane, a zgłoszenie usunięte z bazy.

**Karta wymagań „Przypomnienie hasła”**

Użytkownik po naciśnięciu odnośnika zapomniałem hasła, zostaje przekierowany do formularza przypomnienia hasła:



The screenshot displays a web application interface. At the top, there is a dark grey header bar containing four elements: a button labeled 'ZALOGUJ' with the subtext 'lub zarejestruj się', a button labeled 'HOME', a text input field labeled 'Gdzie, co?', and a button labeled 'SZUKAJ'. Below the header, a map of Romania is visible, showing various towns and roads. Overlaid on the map is a white modal window. The modal window has a title bar that says 'login / email:'. Below the title bar is a text input field. At the bottom of the modal window is a button labeled 'Przypomnij!'. The modal window also has a close button (an 'X' icon) in the top right corner.

Rysunek 2.4: Formularz przypomnienia hasła.

**Testy akceptacyjne „Przypomnienie hasła”**

- Po wpisaniu istniejącego loginu lub maila, na maila wysyłane jest zapytanie, czy chcemy resetować hasło z linkiem aktywującym. Po jego kliknięciu otrzymujemy drugi mail z wygenerowanym nowym hasłem, które później możemy zmienić na zakładce „Ustawienia”.
- Użytkownik po podaniu błędnego loginu/maila zostaje o tym poinformowany ekranem:

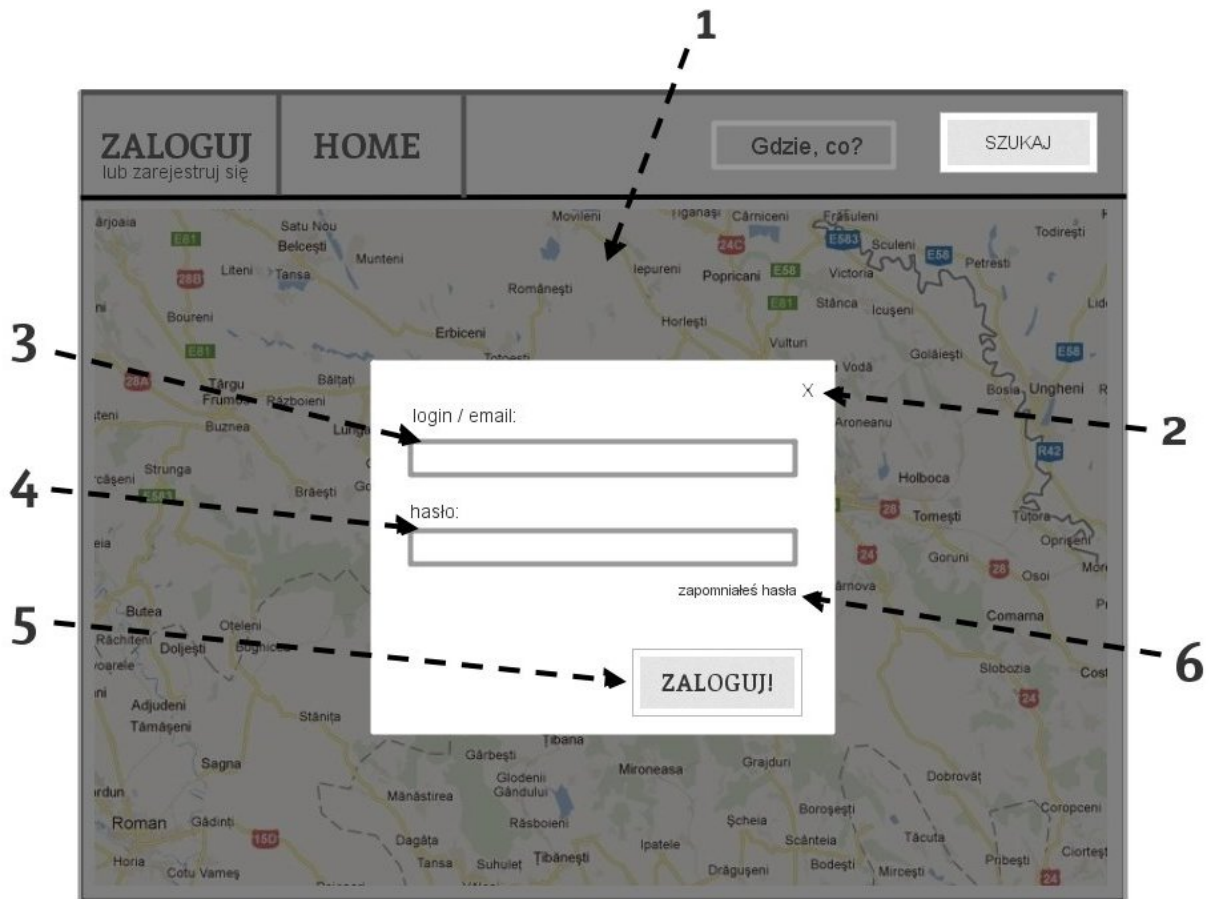




Rysunek 2.5: Ekran informacji o błędnym wypełnieniu loginu/hasła.

**Karta wymagań „Logowanie użytkownika”**

Użytkownik po naciśnięciu odnośnika zaloguj, zostaje przekierowany do formularza logowania:



Rysunek 2.6: Formularz logowania.

1. reszta strony wyszarzona, formularz logowania jako „overlay”
2. klikając X’a wracamy do wyszarzonej strony
3. miejsce na wpisanie loginu lub email
4. miejsce na wpisanie hasła
5. link do formularza przypomnienia hasła
6. przycisk wysyłający formularz logowania

**Testy akceptacyjne „Logowanie użytkownika”**

- Użytkownik tylko po podaniu istniejącego loginu oraz pasującego do niego hasła zostaje zalogowany i przeniesiony na oglądaną przed logowaniem stronę.
- Użytkownik po zalogowaniu ma dostęp do dodatkowych podstron – Ustawienia, Podziel się!
- Użytkownik po podaniu błędnego hasła lub nieistniejącego loginu, zostaje o tym poinformowany ekranem:



Rysunek 2.7: Ekran informacji o błędnym hasle lub loginie.

1. informacja o typie błędu
2. oznaczenie pól jako błędnie wypełnionych, pole hasła pozostaje puste, pole loginu zostaje jako ostatnia wpisana wartość

- Użytkownik po zalogowaniu widzi zmodyfikowane menu – na wszystkich podstronach serwisu.





Rysunek 2.8: Zmodyfikowane menu na górze

1. avatar pobrany z gravatar.com oraz nazwa zalogowanego użytkownika
2. wpis w menu przenoszący do podstrony ustawień zalogowanego użytkownika
3. wpis w menu przenoszący do podstrony udostępniania streamu audio/video

**Karta wymagań „Udostępnianie streamu audio/video krok pierwszy”**

Użytkownik po wybraniu z menu opcji podziel się, zostaje przekierowany do formularza udostępniania audio/video:

The diagram shows a web form titled "Podziel się streamem!". It contains the following elements:

- 1**: Points to the "Kategoria:" label.
- 2**: Points to the "Tytuł:" label.
- 3**: Points to the "Kamera:" dropdown menu.
- 4**: Points to the "Podziel się!" button.
- 5**: Points to the "Dostępność:" dropdown menu.

The form includes a list of categories with checkboxes: SPORT, MUZYKA, GRA, and FILM. It also has input fields for "Tytuł:", "Kamera:", "Jakość:", and "Dostępność:". At the bottom are two buttons: "Anuluj" and "Podziel się!".

Rysunek 2.9: Formularz udostępniania audio/video.

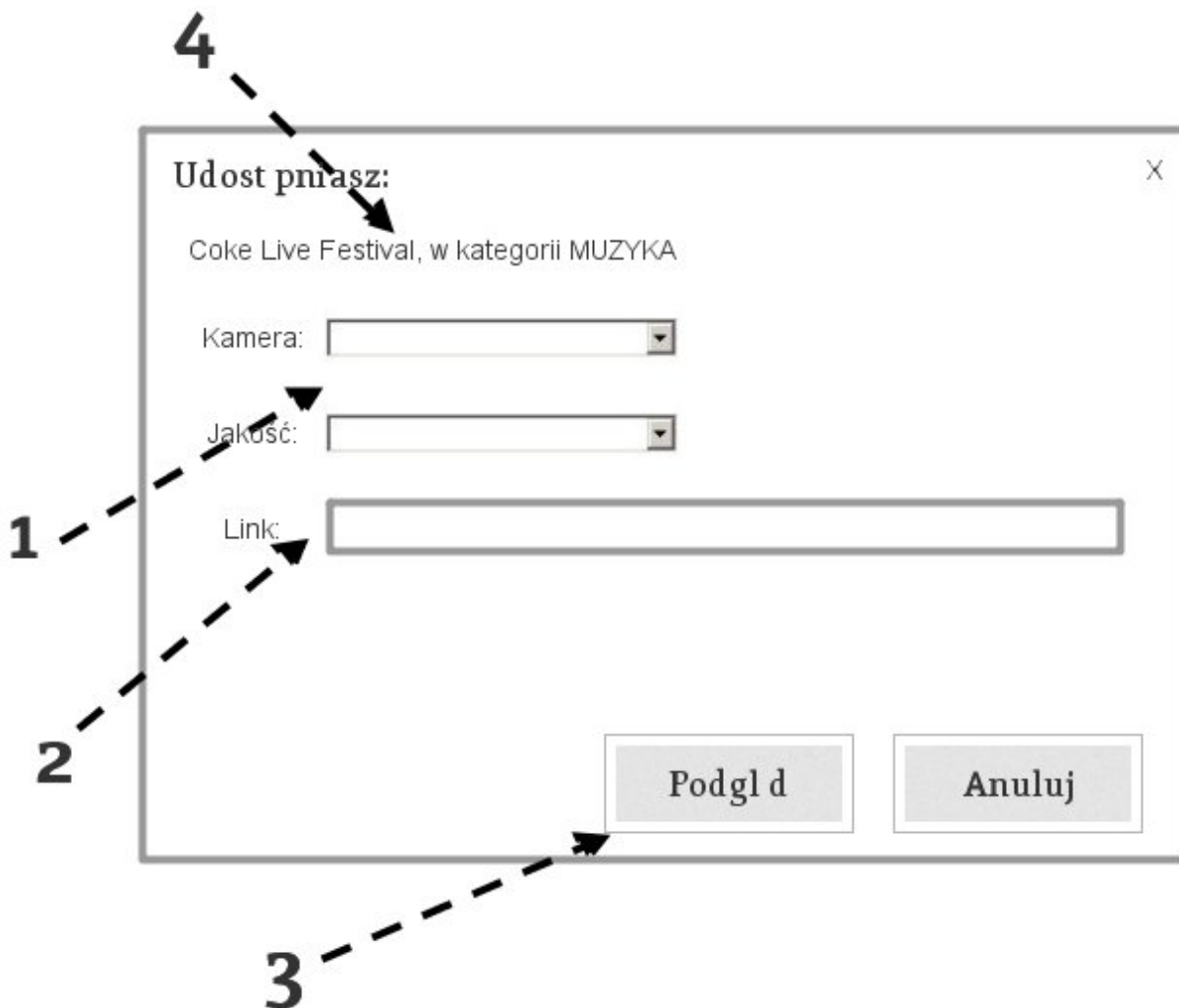
1. Aby udostępnić stream należy wybrać kategorię, do której ten stream ma być przypisany. Lista kategorii jest przykładową listą, administrator ma mieć możliwość modyfikowania tej listy z poziomu panelu administracyjnego.
2. Jeżeli w systemie istnieje więcej niż jedna kamera, tutaj wybierana jest kamera źródłowa. Domyślnie wybraną jest pierwsza systemowa kamera.
3. Możemy wybrać jakość transmisji spośród:  
**Wysoka** Wysoka jakość z preferencją wideo nad audio.  
**Średnia** Średnia jakość zarówno wideo jak audio.  
**Niska** Niska jakość wideo z preferencją jakości audio nad wideo
4. Po kliknięciu „Podziel się” na formularzu udostępniania audio/video, użytkownikowi otwiera się nowe okno przeglądarki gdzie prezentowane są dane udostępnionego streamu (krok drugi). W starym oknie użytkownik zostaje przekierowany do poprzednio odwiedzanej strony – przed wybraniem „Podziel się”.
5. Wybór dostępności streamu, albo publiczny – wszyscy użytkownicy systemu mogą go oglądać/udostępniać (domyślna opcja) – lub prywatny – tylko posiadający link do streamu mogą go zobaczyć.

**Testy akceptacyjne „Udostępnianie streamu audio/video krok pierwszy”**

- Jeżeli w systemie nie ma kamery, użytkownik jest informowany o braku możliwości udostępniania, przycisk „Podziel się!” staje się nieaktywny.

**Karta wymagań „Udostępnianie streamu audio/video krok drugi”**

Po kliknięciu „Podziel się” na formularzu udostępniania audio/video, użytkownikowi otwiera się nowe okno przeglądarki gdzie prezentowane są dane udostępnionego streamu. W starym oknie użytkownik zostaje przekierowany do poprzednio odwiedzanej strony – przed wybraniem „Podziel się”.



Rysunek 2.10: Okno danych udostępnianego streamu audio/video.

1. możliwość zmiany kamery oraz jakości w trakcie nadawania
2. jeżeli wcześniej wybrano publiczną dostępność streamu, wyświetlamy tutaj unikalny link do dzielenia się streamem ze znajomymi np. za pomocą Facebook'a, w przeciwnym wypadku wyświetlamy „permalink” streamu w formie `http://domena.com/login_usera/slug_tytulu_streamu`
3. przycisk otwierający w nowym oknie link streamu
4. informacja na temat tytułu oraz kategorii udostępnianego streamu

**Testy akceptacyjne „Udostępnianie streamu audio/video krok drugi”**

- Po kliknięciu na pole tekstowe linku, zaznacza się cała zawartość. Dodatkowo zaznaczona zawartość zostaje skopiowana do schowka, na ekranie pojawia się informacja „Link skopiowano do schowka!”

## 2.2. Analiza wymagań i dobór rozwiązań

Po rozmowie z klientem, na podstawie głównego zadania systemu oraz innych przygotowanych kart wymagań, można przejść do fazy analizy wymagań więcej w w rozdziale 1.3.7 Zwinna analiza wymagań.

Celem tego etapu jest stworzenie wstępnej architektury systemu, przetestowanie wydajności technologii – najczęściej przy użyciu prototypu systemu.

Jako początek rozważań architektury systemu skorzystano z elementów metody Architecture Tradeoff Analysis Method – Drzewo użyteczności – szczegóły metody można poznać w [KKP00].

Pełna metoda ATAM wprowadza zbyt dużą formalizację i generuje zbyt dużą ilość dokumentacji, z której metodyki zwinne chcą rezygnować. Dodatkowo zgodnie z założeniami XP (szczegóły w rozdziale 1.3.1 Założenia), wszystkie ustalenia sporządzone w tej fazie mogą ulec zmianie na każdym późniejszym etapie prowadzenia projektu. Zwinna analiza wymagań nie zamyka możliwości późniejszej zmiany technologii, architektury czy dowolnego z elementów wspomagających projekt.

### 2.2.1. Drzewo użyteczności (ATAM Usability Tree)

Niżej umieszczono *Drzewo Użyteczności* metody ATAM w formie listy. Pokazuje w bardzo przejrzysty sposób najważniejsze aspekty architektury systemu.

- **WYDAJNOŚĆ**

- Opóźnienie transmisji danych
  - \* opóźnienie wideo dostosowane do aktualnej przepustowości łącza, jak najmniejsze
  - \* brak lub minimalne opóźnienia transmisji audio
- Ilość użytkowników
  - \* Liniowa zależność ilość użytkowników używających system od zużywanych zasobów
  - \* Minimalna ilość użytkowników aktywnie poruszających się po stronie – 1000 osób
  - \* Minimalna ilość użytkowników oglądających jeden strumień wideo – 100 osób

- **MODYFIKOWALNOŚĆ**

- Zarządzanie użytkownikami
- Zarządzanie dostępnymi streamami
- Zarządzanie kategoriami streamów
- Aktualizacja i dystrybucja oprogramowania mobilnego
- Łatwa modyfikacja wyglądu strony

- **DOSTĘPNOŚĆ**

- Dostępność całej usługi na poziomie 99,9% rocznie
- Dostępność usługi na urządzenia mobilne (tablety, telefony)
- Dostępność usługi na desktopy

- **SKALOWALNOŚĆ**

- Odporność na geometryczną „nagłą popularyzację”
- Liniowa zależność ilość użytkowników/zużywane zasoby

- **BEZPIECZEŃSTWO**

- System logowania

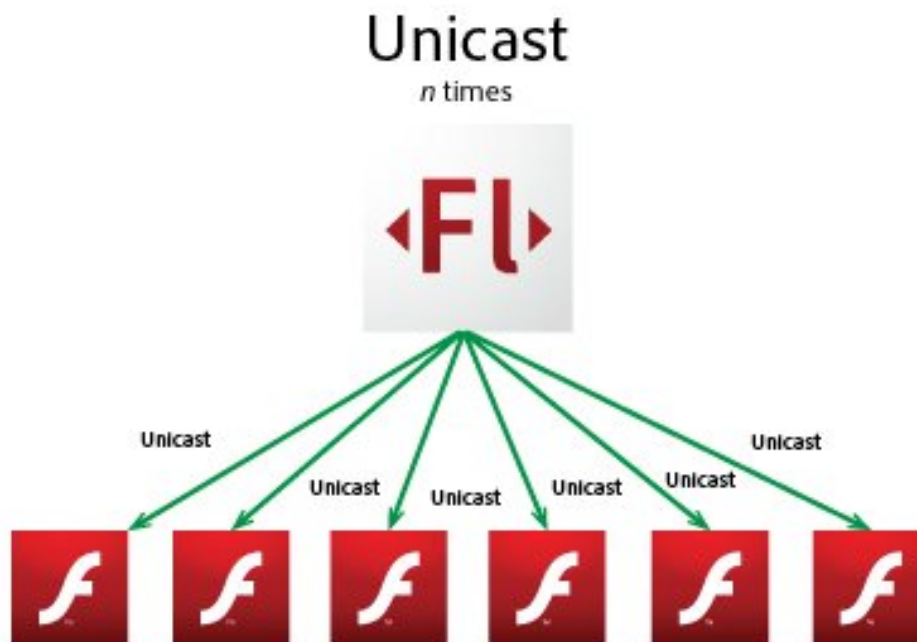
- Stream publiczny/prywatny
- Szyfrowanie
  - \* szyfrowanie transmisji wideo
  - \* szyfrowanie transmisji audio
  - \* szyfrowanie transmisji http i każdej innej transmisji danych poza system

### 2.2.2. Wstępna architektura systemu

Docelowy system będzie składał się z **Interfejsu WWW**, odpowiedzialnego za prezentację mapy i dostosowywanie się do zmiennej szerokości ekranu urządzeń. Wszystkie streamy będą umieszczone w relacyjnej **Bazie danych** z rozszerzeniem GIS, dzięki czemu możliwe będzie łatwe zapisanie lokalizacji streamu.

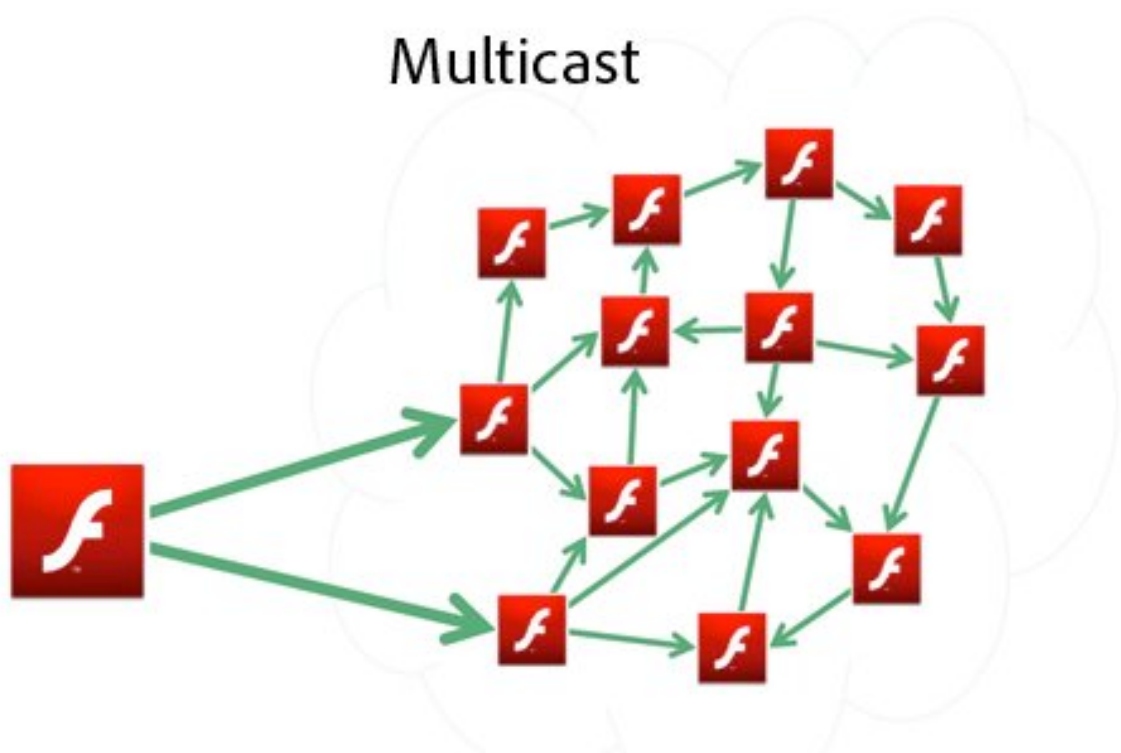
Po stronie komputerów stacjonarnych, za pobieranie obrazu z kamery i wyświetlanie obrazu u drugiego użytkownika będzie odpowiedzialny obiekt SWF **Adobe Flash Player**. W przypadku urządzeń mobilnych, do przygotowania aplikacji obsługującej zarówno iPhone'a jak i Androida wykorzystany zostanie otwarty framework Adobe Flex i jego mobilne wsparcie w postaci **Adobe AIR for Mobile** (alternatywne rozwiązania, które rozważano, opisane są w rozdziale 2.2.3 *Technologia*.)

Do rozproszenia ruchu i stworzenia systemu niezależnego od pojedynczego punktu połączenia sieciowego – osoby udostępniającej stream – będą wykorzystane sieci ze wspomaganiem równorzędnym wbudowane w Adobe Flash Platform. Zasada działania sieci P2P wymusza udostępnianie oglądanego streamu, tak aby zapewnić prostą zależność – im więcej użytkowników chce oglądać, tym więcej użytkowników udostępnia żądany stream. Firma Adobe nazywa tę technologię **Multicast/P2P Multicast**, niżej przedstawione są schematy pokazujące różnicę pomiędzy transmisją Unicast, a P2P Multicast.



Rysunek 2.11: Przedstawienie transmisji Unicast w Adobe Flash Platform, obrazek pochodzi z [Krc10].

Jak widać w tym przypadku potrzebny jest serwer z łączem o odpowiedniej przepustowości, aby zapewnić ciągłość transmisji. Klienci biorący udział w transmisji nie komunikują się w żaden sposób ze sobą.

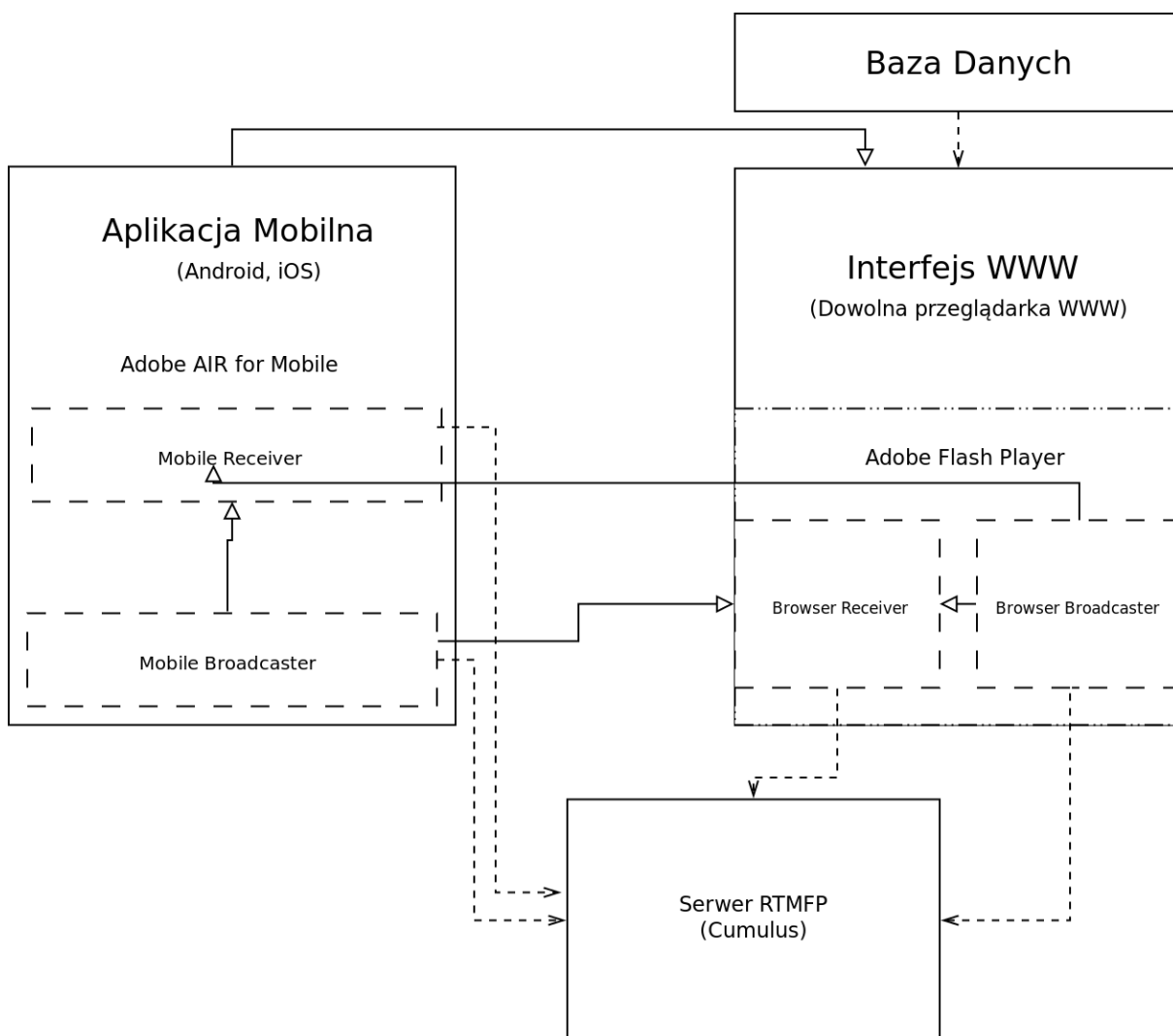


Rysunek 2.12: Przedstawienie transmisji P2P Multicast w Adobe Flash Platform, obrazek pochodzi z [Krc10].

W przypadku transmisji P2P Multicast łączy się obciążone względem swoich możliwości. Obciążenie rozłożone na całość sieci ze wspomaganiem równorzędnym. Klienci komunikują się ze sobą współdzieląc zasób.

Decyzja o wykorzystaniu technologii Adobe Flash Platform miała bezpośredni wpływ na architekturę systemu. Alternatywy, które rozważano opisane są w rozdziale 2.2.3 Technologia.

Niżej przedstawiono schemat architektury pokazujący powiązanie pomiędzy poszczególnymi, wcześniej wymienionymi, elementami systemu.



Rysunek 2.13: Wstępna architektura systemu.

**Interfejs WWW** Centralny interfejs użytkownika z systemem, umożliwia przeglądanie streamów z poziomu przeglądarki WWW, implementuje design dostosowywany do szerokości ekranu urządzenia na którym się go przegląda. Współpracuje z wszystkimi nowoczesnymi przeglądarkami WWW. Udostępnia API do komunikacji z aplikacją mobilną czy samym obiektem SWF.

**Baza Danych** Centralne miejsce składowania wszystkich danych przechowywanych w systemie. Umożliwia szybkie wyszukiwanie, jest bezpośrednio połączona z Interfejsem WWW, który przez swoje API udostępnia przechowywane przez nią dane.

**Browser Receiver** Aplikacja Adobe Flash Player, umożliwiającą dołączenie do streamu wideo nadawanego przez Browser Broadcaster lub Mobile Broadcaster.

**Browser Broadcaster** Aplikacja Adobe Flash Player, umożliwiającą nadawanie streamu wideo bezpośrednio z przeglądarki.

**Mobile Receiver** Aplikacja mobilna Adobe AIR for Mobile, umożliwiającą odbieranie streamu wideo nadawanego przez Mobile Broadcaster lub Browser Broadcaster z poziomu urządzenia wspierającego iOS lub Androida.



**Mobile Broadcaster** Aplikacja mobilna Adobe AIR for Mobile, umożliwiająca nadawanie streamu wideo z poziomu urządzenia wspierającego iOS lub Androida.

**Serwer RTMFP** Serwer przechowujący informacje na temat listy osób biorących udział w transmisji wideo streamów. Dzięki niemu pokonujemy niefiltrowany NAT. Łączą się z nim wszystkie aplikacje Broadcaster oraz Receiver.

### 2.2.3. Technologia

W tym rozdziale umieszczono informacje o tym jakie technologie oraz dlaczego zostały wybrane do realizacji poszczególnych elementów systemu. Przed przedstawieniem wybranej technologii rozważono alternatywy jeżeli takowe istniały.

#### Adobe Flash Platform

Przed wybraniem tej technologii rozważano alternatywne rozwiązania w postaci dwóch technologii. Niżej przedstawiono je wraz z krótkim opisem wyjaśniającym dlaczego nie znalazły zastosowania w systemie.

**Widget JAVA w przeglądarce** Java jest środowiskiem programistycznym dostępnym na każdą platformę. Udo-  
stępnia możliwość uruchomienia widgetów w obrębie przeglądarki, umożliwia wykonywanie połączeń UDP  
oraz TCP. Nie znaleziono biblioteki umożliwiającej stworzenie jednego kodu dla urządzeń iOS oraz An-  
droid. Dodatkowo wykorzystanie JAVY wymusiłoby samodzielną implementację P2P w transmisji wideo  
z poziomu przeglądarki. Ostatecznym „gwoździem do trumny” tej technologii jest mała popularność oraz  
najczęściej blokowanie Javy w przeglądarkach.

**Websockets** Dostępne w specyfikacji HTML5 (<http://tools.ietf.org/html/rfc6455>), umożliwia-  
jące nawiązywanie połączeń TCP pomiędzy serwerem, a przeglądarką. Aktualnie wspierane przez Firefox  
4+, Google Chrome 16+, Opera 11+ oraz Safari 5+. Powodem na rezygnację z tej technologii jest brak  
obsługi pobierania strumienia wideo z kamery z poziomu przeglądarki, brak przykładów lub biblioteki po-  
święconej zagadnieniu P2P wykorzystującej websockets oraz brak wsparcia technologii dla przeglądarek  
mobilnych.

Platforma technologii Flash składająca się z kilku  
elementów. Przy odpowiednim ich wykorzystaniu  
umożliwia stworzenie aplikacji działających praktycz-  
nie na dowolnej platformie i dowolnej przeglądarce.

**Adobe Flex** Framework z otwartym kodem źródło-  
wym umożliwiający tworzenie aplikacji w śro-  
dowiskach Adobe AIR oraz Adobe Flash Player.

**Adobe Flash Player** Środowisko dostępne we  
wszystkich desktopowych przeglądarkach i  
niektórych mobilnych (Android). Służy do od-  
twarzania animacji SWF. Pozwala wykorzystać  
szereg klas Flex’a niebędących częścią Adobe  
AIR.



Rysunek 2.14: Logo Adobe Flash Platform, pobrane z  
Google Images

**Adobe AIR** Środowisko dostępne na szereg platform (Windows, Android, iOS) umożliwiające tworzenie aplikacji mobilnych jak i desktopowych niezależnie od platformy docelowej.

Wybranie tej technologii było koniecznością już na samym początku – stąd jej obecność w architekturze systemu. Wynika bezpośrednio ze specyfikacji wymagań. Jest to jedyne rozwiązanie umożliwiające aktualnie:

- Nadawanie „peer to peer” streamu wideo z przeglądarki/urządzenia mobilnego.
- Odbiór transmisji „peer to peer” streamu wideo z przeglądarki/urządzenia mobilnego.
- Nadawanie „peer to peer” streamu wideo z urządzenia mobilnego.
- Odbiór transmisji „peer to peer” streamu na urządzeniu mobilnym.
- Skalowalność na poziomie miliona klientów w sieci „peer to peer” dla pojedynczego streamu (więcej w [Kau09]).
- Tworzenie aplikacji na iOS oraz Androida z jednego źródła.

Jak każda technologia Adobe Flash Platform ma swoje wady. Nijżej przedstawiono listę najważniejszych z nich.

- Komunikacja losowym, wysokim portem UDP może generować problemy z korporacyjnymi firewallami.
- Wymagana instalacja wtyczki do przeglądarki (w Google Chrome/Chromium jest instalowana automatycznie alternatywna wersja).
- Ograniczony support Adobe Flash Player na urządzeniach firmy Apple.
- W przypadku programów IPA dla urządzeń firmy Apple, aplikacja wykorzystująca Adobe AIR for Mobile dołącza wykorzystywane biblioteki Adobe AIR do samej aplikacji, umożliwiając w ten sposób jej uruchomienie na ww. urządzeniach.

## Cumulus

Darmowa oraz otwarta, bo udostępniona na licencji GPL, implementacja serwera RTMFP (Real Time Media Flow Protocol). W systemie jest on nieodzowny do odkrywania listy potencjalnych peerów posiadających źródło streamów. Jego wykorzystanie jest praktycznie „wymuszone” przez preferencję rozwiązań Open Source nad komercyjnymi rozwiązaniami np. w postaci Adobe Flash Media Interactive Server (<https://github.com/OpenRTMFP/Cumulus>).

Przed wybraniem Cumulusa jako serwera RTMFP brano pod uwagę następujące alternatywy.

**Adobe Cirrus** Darmowy i ograniczony dostęp do Adobe Cirrus za pomocą klucza developerskiego (<http://labs.adobe.com/technologies/cirrus/>). Nie nadaje się do wykorzystania komercyjnego, nie skaluje się ze względu na stały limit wywołań.

**Adobe Flash Media Interactive Server 4.5** Jedyną, lecz także poważną wadą tego rozwiązania jest koszt około 5 900 EURO za licencję.

Poniżej przedstawiono zalety Cumulus, przez które zdecydowano o jego zastosowaniu przy tworzeniu implementacji systemu.

- Wbudowana obsługa P2P „rendez-vous”, czyli zbierania i udostępniania informacji na temat dostępnych peerów dla danego streamu.
- Możliwość rozsyłania streamu wideo na żywo.
- Otwartość oraz darmowość – jedyna otwarta implementacja serwera RTMFP.
- Możliwość pisania rozszerzeń serwera za pomocą języka LUA.
- Wbudowana obsługa komunikacji AMF (Action Message Format): pull, push, RPC.
- Przewidziana skalowalność oraz zarządzanie obciążeniem za pomocą CumulusEdge.
- Łatwa konfiguracja oraz instalacja.
- Wieloplatformowość.
- Aktywna grupa dyskusyjna.

Jak każda z technologii, także i Cumulus ma swoje wady. Niżej przedstawiono ich listę.

- Brak wbudowanej obsługi „zwykłego” protokołu RTMP (Real Time Messaging Protocol), jako alternatywnego źródła streamu w przypadku blokady firewall po stronie klienta.

## Django

Framework napisany w Pythonie, umożliwiający proste i szybkie tworzenie stron internetowych. W systemie stworzony zostanie za jego pomocą Interfejs WWW, będący centrum wymiany informacji i możliwości przeglądania streamów. Strona domowa projektu to <http://djangoproject.com>.

Przed wybraniem Django, jako technologie do wykorzystania brano pod uwagę następujące alternatywy.

**Zend** Framework napisany w PHP, technologia nie będąca bliska autorowi pracy (<http://framework.zend.com>).

**Pyramid** Framework napisany w Pythonie, mniej popularny od Django. Brak doświadczenia w wdrażaniu aplikacji Pyramid (<http://www.pylonsproject.org>).

**Rails** Framework napisany w Ruby. Brak doświadczenia w Ruby oraz w wdrażaniu aplikacji Rails (<http://rubyonrails.org>).

Wykorzystując konkurencyjne frameworki dało by się również zrealizować Interfejs WWW. Poniżej przedstawiono zalety Django, przez które zdecydowano o jego zastosowaniu przy tworzeniu implementacji systemu.

- Znajomość procesu wdrażania oraz doświadczenie w zarządzaniu aplikacjami Django autora pracy.
- Wbudowany/automatyczny panel admina.
- Wbudowany ORM dla PostgreSQL, MySQL, SQLite.
- Wbudowane wsparcie dla baz danych z rozszerzeniem GIS – geodjango.
- Wbudowana obsługa testów jednostkowych.
- Obszerna baza bibliotek Pythona – np. PyAMF do połączeń Adobe Flash / Django; Fabric do deployowania gotowej aplikacji.
- Tworzony z myślą, DRY – Don't Repeat Yourself.
- Bardzo dobra dokumentacja.
- Napisany w Pythonie.
- Skalowalny do rozmiarów Disqus, Instagram czy Pinterest.



Rysunek 2.15: Logo Django, pobranie ze strony projektu.

Jak każda z technologii, także i Django ma swoje wady. Niżej przedstawiono ich listę.

- Wymaga odpowiedniego hostingu. Wymaga wiedzy i doświadczenia we wdrażaniu.
- Konieczność poznania Pythona.

## jQuery

Biblioteka JavaScript tworząca wspólne API dla wszystkich przeglądarek służące do: przechodzenia i wybierania elementów z drzewa dokumentu HTML, tworzenia animacji, interakcji AJAX oraz zarządzania zdarzeniami. W systemie jest dokładnie w tych celach wykorzystywana (<http://jquery.com>).

Przed wybraniem jQuery, jako technologie do wykorzystania brano pod uwagę następujące alternatywy.

**mootools** Jest frameworkiem JavaScriptowym, jest narzędziem zbyt poważnym do zastosowania w systemie (<http://mootools.net>).

**prototypeJS** Jest podobnie jak jQuery biblioteką JavaScript, mniejsza od mootools, ale wolniejsza od jQuery (<http://prototypejs.org>).

**dojotoolkit** Jest modularnym frameworkiem JavaScriptowym, zbyt poważnym do zastosowania w systemie (<http://dojotoolkit.org>).

Poniżej przedstawiono zalety jQuery, przez które zdecydowano o jego zastosowaniu przy tworzeniu implementacji systemu.

- Zajmuje jedynie 31KB.
- Zgodny z CSS3.
- API wspólne dla wszystkich przeglądarek: IE 6.0+, FF 10+, Safari 5.0+, Opera, Chrome.
- Dobra dokumentacja i przykłady.
- Duża popularność.



Rysunek 2.16: Logo jQuery, pobranie ze strony projektu.

Jak każda z technologii, także i jQuery ma swoje wady. Niżej przedstawiono ich listę.

- Dalej jest to ten sam pocziwy JavaScript.
- Nie jest to framework, a jedynie biblioteka.

## Bootstrap

Framework do tworzenia interfejsu WWW, napisany w LESS, kompilujący się do CSS. Dodatkowo za pomocą jQuery, umożliwia skorzystanie z wielu niestandardowych komponentów, czy też animacji. Za jego pomocą zostanie stworzony layout na potrzebny system, dostosowany do obsługi urządzeń mobilnych zachowując przy tym jedno źródło (<http://twitter.github.com/bootstrap>).

Nie znaleziono alternatywnej technologii, która mogła by być porównana na każdej płaszczyźnie z Bootstrapem. Większość rozwiązań trzeba rozszerzać lub kompletować o to co jest gotowe i wbudowane w Bootstrap.

Poniżej przedstawiono zalety Bootstrapa, przez które zdecydowano o jego zastosowaniu przy tworzeniu implementacji systemu.

- Wbudowana obsługa responsywności – klasy CSS zależne do szerokości ekranu.
- Wbudowany reset stylów CSS, tak aby przeglądarki wyświetlały w taki sam sposób wszystkie komponenty HTML'a (marginesy, paddingi).
- Formatowanie domyślne standardowych tagów HTML.
- Wbudowane klasy pomocnicze do typografii.
- Wbudowane komponenty rozszerzające tagi HTML.
- Wbudowany, dwunasto-kolumnowy grid.
- Rozszerzalny dzięki jQuery o dodatkowe możliwości (animacje, dodatkowe komponenty).
- Zbudowany za pomocą LESS (<http://lesscss.org>) – duże ułatwienie przy pisaniu kodu CSS m.in. poprzez: zmienne, zagnieżdżenia oraz „mix-iny”.
- Bardzo dobra dokumentacja oraz przykłady.



Rysunek 2.17: Logo Bootstrap, pobranie ze strony projektu.

Jak każda z technologii, także i Bootstrap ma swoje wady. Niziej przedstawiono ich listę.

- Aby w pełni skorzystać trzeba znać CSS i poznać LESS.
- Pliki źródłowe LESS trzeba kompilować do CSS, co jest dosyć uciążliwe.
- Raz wykorzystany stanie się podstawą przy większości projektów webowych.

## FlashDevelop

Jedyne otwarte i darmowe IDE współpracujące z Adobe Flash Platform (wykorzystujące SDK Flex'a), umożliwiające tworzenie aplikacji z wykorzystaniem Adobe Flash Platform. Wszystkie części systemu związane z technologią Adobe zostaną przygotowane za pomocą tego IDE (<http://flashdevelop.org>).

Nie znaleziono alternatywnego rozwiązania darmowego pozwalającego na porównanie z FlashDevelop. Flash Builder Premium 4.6, jest oryginalnym i polecanym przez Adobe rozwiązaniem. Koszt to „jedyne” 639 EURO za pełną licencję jednostanowiskową.

Poniżej przedstawiono zalety FlashDevelop, przez które zdecydowano o jego zastosowaniu przy tworzeniu implementacji systemu.

- Otwartość oraz darmowość.
- Gotowe szablony aplikacji Adobe Flex/AIR/AIR for Mobile.
- Zaawansowane podpowiadanie kodu („code completion”).
- Możliwość pobrania i instalacji Adobe Flex SDK z poziomu aplikacji.
- Wbudowany Flash debugger.



Rysunek 2.18: Logo FlashDevelop, pobranie ze strony projektu.

Jak każde z rozwiązań, także i FlashDevelop ma swoje wady. Niżej przedstawiono ich listę.

- Dostępna jest jedynie wersja na platformę Windows.
- Szcątkowa dokumentacja.
- Wymaga podstawowej znajomości technologii Adobe, aby odnaleźć się w interfejsie i zacząć projekt.

## PostgreSQL

Serwer obiektowo-relacyjnej bazy danych, umożliwiający przechowywanie i szybki dostęp do wszelkich danych. Jest rozwijany od ponad 15 lat. Jego sprawdzona przez ten czas architektura potwierdza jego opinię, jako serwera niezawodnego, utrzymującego spójność i poprawność danych. W projekcie jest podstawowym mechanizmem składowania danych (<http://www.postgresql.org>).

Przed wybraniem PostgreSQL, jako technologie do wykorzystania brano pod uwagę następujące alternatywy.

**MySQL** Najpopularniejsza relacyjna baza danych. Posiada wbudowane rozszerzenie GIS, jest wspierana przez ORM Django. Szybka, lecz nie tak niezawodna jak PostgreSQL. (<http://www.mysql.com>).

**SQLite** Relacyjna baza danych umożliwiająca przechowywanie bazy w pliku. Posiada ograniczone transakcje oraz rozszerzenie GIS, jest wspierana przez ORM Django. Mało wydajna dla aplikacji wielowątkowych (<http://www.sqlite.org>).

Poniżej przedstawiono zalety PostgreSQL, przez które zdecydowano o jego zastosowaniu przy tworzeniu implementacji systemu.

- Otwartość oraz darmowość.
- Niezawodność, stabilność.
- Duża wydajność.
- Wbudowane narzędzia do tworzenia backupu danych – `pg_dump`.
- Dostępne narzędzia do replikacji oraz zarządzania obciążeniem np. `pgpool2` – ważne w przypadku skalowalności.
- Dostępne rozszerzenie GIS (Geographic Information System) w postaci PostGIS (<http://postgis.refractory.net>).
- Możliwość uruchomienia na wielu platformach.



Rysunek 2.19: Logo PostgreSQL, pobranie ze strony projektu.

Jak każde z rozwiązań, także i PostgreSQL ma swoje wady. Niżej przedstawiono ich listę.

- Stosunkowo długie otwieranie nowego połączenia.
- Mniejsza popularność i dostępność w pakietach hostingowych niż MySQL'a.



### Google Maps Javascript API

Jest darmową biblioteką Javascript pozwalającą na umieszczenie Mapy Google na własnej stronie internetowej. Dostarcza API zawierające szereg metod pozwalających na zarządzanie obiektem mapy oraz na dodawanie zawartości do Map Google. Łączy się też z wieloma darmowymi usługami dostępnymi od firmy Google (<https://developers.google.com/maps/documentation/javascript>).

W projektowanym systemie będzie ona wykorzystywana do prezentacji streamów posiadających lokalizację oraz umożliwiała wyszukiwanie informacji.

Przed wybraniem Google Maps Javascript API, jako technologie do wykorzystania brano pod uwagę następujące alternatywy.

**OpenLayers** Całkowicie darmowa i otwarta biblioteka wraz z serwerem map, umożliwiająca wyświetlanie interaktywnej mapy na stronie internetowej. Wolniejsza i z mniej dopracowanym interfejsem względem rozwiązania od Google (<http://openlayers.org>).

**Nokia Maps API for JavaScript** Zaawansowana biblioteka JavaScript do wyświetlania map Ovi. Stosunkowo nieduże ograniczenia jeżeli chodzi o ilość zapytań – 1 milion w wersji Trial. Reszta opcji płatna lub do uzgodnienia z Nokia (<http://api.maps.nokia.com/en/overview.html>).

**Bing Maps AJAX** Biblioteka JavaScript umożliwiająca skorzystanie z map <http://www.bing.com/maps>. Korzysta z map dostarczanych przez Nokia, udostępniają własne API. Gorszy interfejs oraz aktualność map i zdjęć satelitarnych niż w przypadku rozwiązania od Google (<http://www.bingmapsportal.com/isdk/ajaxv7>).

Poniżej przedstawiono zalety Google Maps Javascript API, przez które zdecydowano o jego zastosowaniu przy tworzeniu implementacji systemu.

- Biblioteka jest darmowa.
- Biblioteka działa z większością używanych dzisiaj przeglądarek (IE 7.0+ (Windows), Firefox 3.0+ (Windows, Mac OS X, Linux), Safari 4+ (Mac OS X, iOS), Chrome (Windows, Mac OS X, Linux), Android, BlackBerry 6, Dolfin 2.0+ (Samsung Bada)).
- Biblioteka dostosowana jest do mobilnych urządzeń – zarówno pod względem prędkości jak i wyglądu oraz interfejsu użytkownika.
- Łączy się z usługą wyszukiwania lokalizacji i udostępnia dane dotyczące położenia danej lokalizacji.
- Bardzo dobra dokumentacja.
- Bardzo dobre przykłady wykorzystania API.
- Dokładność map.
- Popularność rozwiązania.

Jak każde z rozwiązań, także i Google Maps Javascript API ma swoje wady. Niżej przedstawiono ich listę.

- Brak usługi geolokalizacji po adresie IP.



Rysunek 2.20: Logo Google Developers, odpowiedzialnego za Google Maps JavaScript API, pobranie ze strony projektu.

### Inne technologie

Przy tworzeniu systemu skorzysta się również z kilku mniejszych narzędzi, czy też bibliotek wartych wymienienia. Niżej przedstawiono ich listę.

**MaxMind GeoIP Lite** Biblioteka udostępniająca bazę IP mapowaną do nazwy lokalizacji i położenia za pomocą prostego API (<http://www.maxmind.com/app/geolite>)

**GIT** Darmowy i otwarty, rozproszony system kontroli wersji (<http://git-scm.com>).

**GitHub** Serwis udostępniający darmowe miejsce na zdalne repozytorium GIT'a – dla projektów o otwartym kodzie źródłowym (<http://github.com>).

**Fabric** Biblioteka i narzędzie linii komend umożliwiające pisanie skryptów automatycznie wykonywanych poprzez SSH na zdefiniowanych hostach. Umożliwia stworzenie procesu automatycznej aktualizacji systemu np. z aktualnego repozytorium (<http://fabfile.org>).

**Factory\_boy** Biblioteka umożliwiająca w prosty sposób tworzenie fabryk do modeli danych. Bardzo pomocna przy tworzeniu testów jednostkowych i generowaniu danych potrzebnych do ich przeprowadzenia ([https://github.com/rbarrois/factory\\_boy](https://github.com/rbarrois/factory_boy)).

**South** Aplikacja Django umożliwiająca proste zarządzanie procesem migracji bazy danych. Dotyczy to zarówno struktury jak i samych danych (<http://south.aeracode.org>).

**Virtualenv** Narzędzie do tworzenia odizolowanych środowisk Python'owych, umożliwia równoległe uruchomienie kilku wersji Pythona z własną kolekcją bibliotek (<http://pypi.python.org/pypi/virtualenv>).

**PIP** Narzędzie linii komend do pobierania i instalowania bibliotek Python'owych. Bardzo przydatne do zarządzania wymaganiami środowiska Python'owego na serwerach wdrożeniowych (<http://pypi.python.org/pypi/pip>).

**Proxmox** Dystrybucja Debiana umożliwiająca wirtualizację na poziomie KVM oraz OpenVZ ([http://pve.proxmox.com/wiki/Main\\_Page](http://pve.proxmox.com/wiki/Main_Page)).

**Ubuntu Server** Jedna z popularniejszych dystrybucji Debiana, w projekcie wykorzystywana jako podstawowa dla wszystkich maszyn wirtualnych (<http://ubuntu.com>).

**Cherokee** Silnik serwera WWW z graficznym interfejsem zarządzania, pozwalający na udostępnianie zawartości statycznej (obrazki, strony HTML), treści dynamicznej, w szczególności Python'a (Django) za pomocą kontenera uWSGI (<http://www.cherokee-project.com>).

**Dia** Otwarte oprogramowanie umożliwiające tworzenie diagramów. Wszystkie diagramy w pracy zostały wykonane za jego pomocą (<http://projects.gnome.org/dia>).

## 3. Implementacja prototypu systemu

Ten rozdział poświęcony jest szczegółom związanym z implementacją prototypu systemu. Była to najbardziej pracochłonna część pracy. Celem implementacji systemu jest sprawdzenie działania technologii jaką jest Adobe P2P Multicasting. Czy rozwiązanie to nadaje się dla projektowanego „Społecznościowego, internetowego systemu monitoringu”.

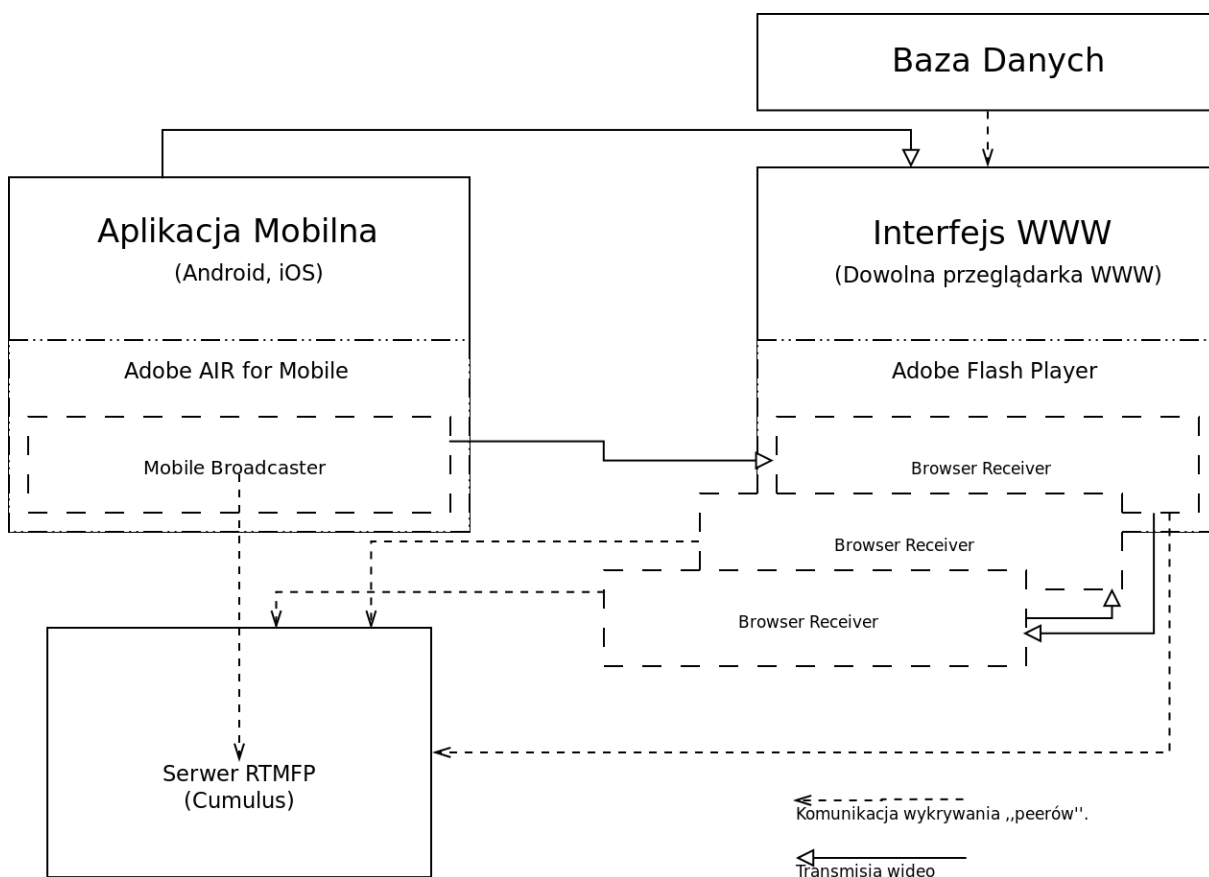
### 3.1. Co zostało zaimplementowane

Prototyp systemu dostępny jest on-line pod adresem <http://facewithme.com>. Przy tworzeniu prototypu systemu skupiono się na pięciu elementach systemu (szczegóły architektury znajdują się w rozdziale 2.2.2 Wstępna architektura systemu):

- Interfejs WWW
- Baza danych
- Serwer RTMFP
- Mobile Broadcaster
- Browser Receiver

Dzięki wybraniu ww. elementów udało się stworzyć system zdolny do przetestowania transmisji wideo pomiędzy Mobile Broadcaster, a Browser Receiver, za pomocą technologii Adobe P2P Multicast. Interfejs WWW umożliwia dodatkowo ładną prezentację oraz wyszukiwanie stream'ów w zadanej lokalizacji czy też personalizację ustawień.

Na poniższym rysunku widać architekturę zaimplementowanej części systemu wraz z prezentacją transmisji P2P. Należy zwrócić uwagę, że Serwer RTMFP nie bierze czynnego udziału w transmisji wideo, służy jedynie do rozgłaszania klientom udostępniających stream. Transmisja wideo odbywa się pomiędzy Mobile Broadcasterem, a klientami Browser Receiver.

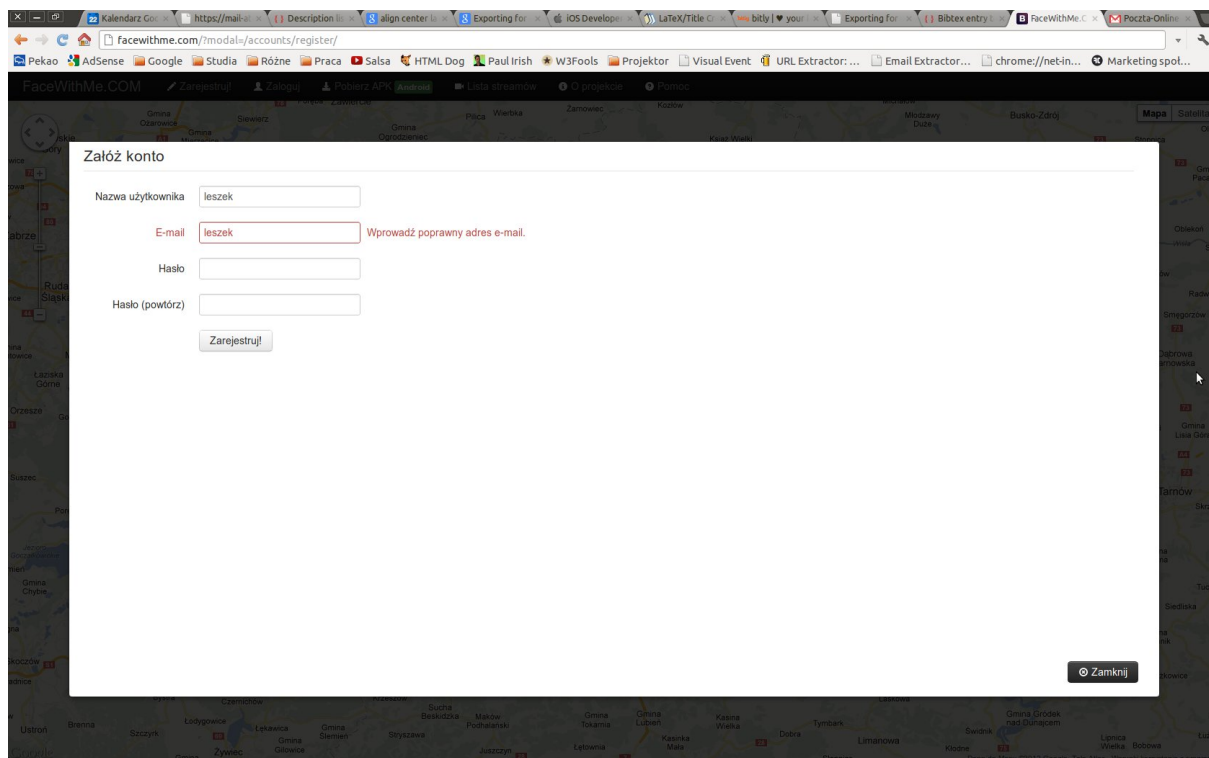


Rysunek 3.1: Architektura implementacji prototypu systemu uwzględniająca transmisję wideo z Mobile Broadcastera do kilku Browser Receiverów.

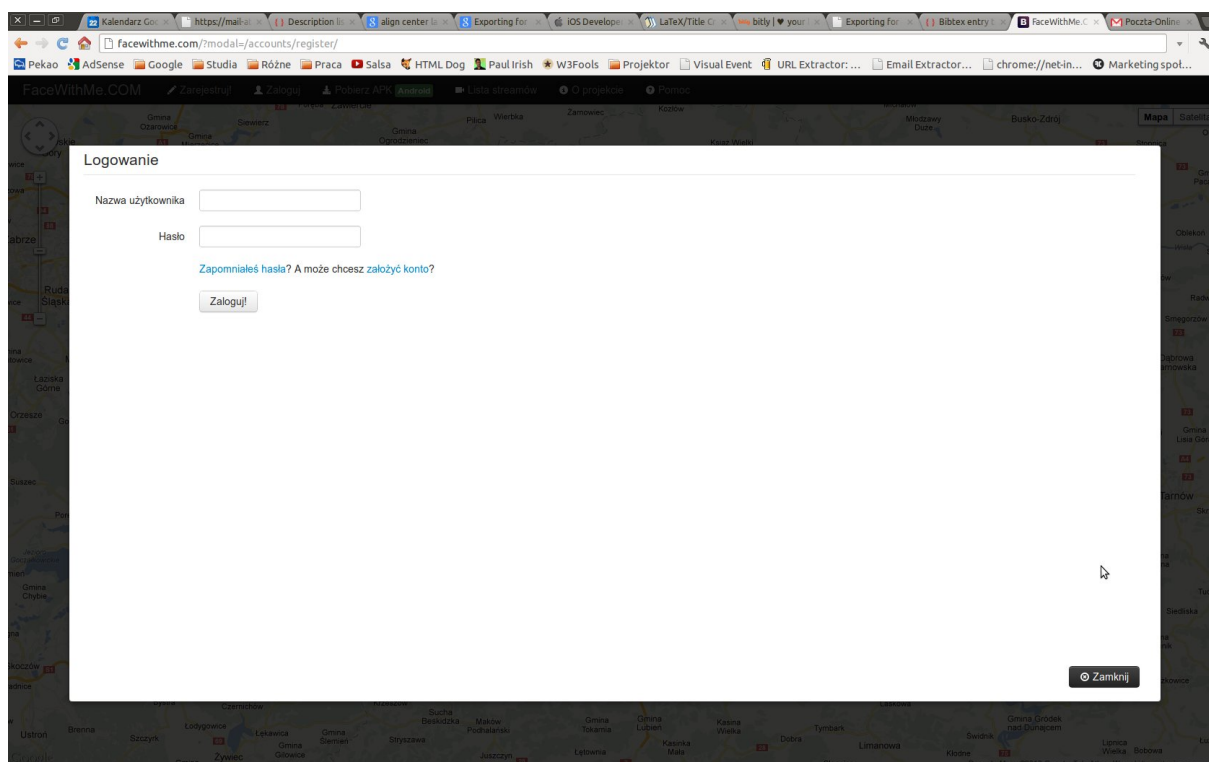
### 3.1.1. Interfejs WWW

Niżej przedstawiono elementy, które udało się zaimplementować w części systemu nazwanej Interfejs WWW. Szczegóły i opis architektury znajdują się w rozdziale 2.2.2 Wstępna architektura systemu.

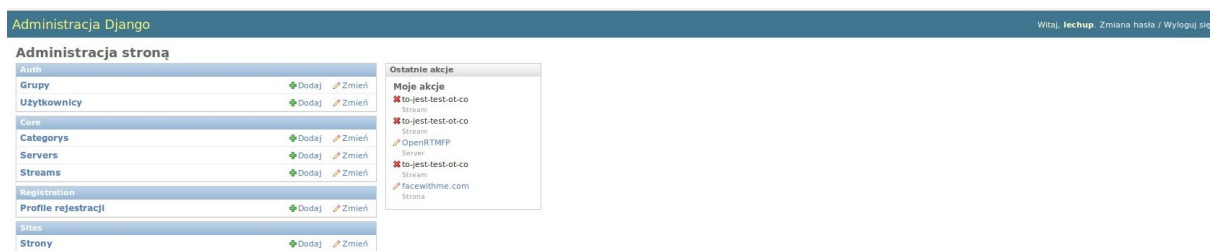
- System rejestracji użytkownika (<http://facewithme.com/accounts/register>), można zobaczyć na rysunku 3.2.
- System logowania użytkownika (<http://facewithme.com/accounts/login>), można zobaczyć na rysunku 3.3.
- Panel administracyjny do zarządzania streamami, kategoriami i użytkownikami (<http://facewithme.com/admin>), można zobaczyć na rysunku 3.4.
- Interaktywną mapę prezentującą streamy (<http://facewithme.com>), można zobaczyć na rysunku 3.5.
- Funkcję automatycznej lokalizacji użytkownika i ustawienie pozycji mapy zgodnie z nią (<http://facewithme.com>), można zobaczyć na rysunku 3.5.
- Listowanie wszystkich streamów nadawanych w systemie z podziałem na kategorie (<http://facewithme.com/stream/list>), można zobaczyć na rysunku 3.6.
- Wyświetlanie Browser Receivera, można zobaczyć na rysunku 3.7.



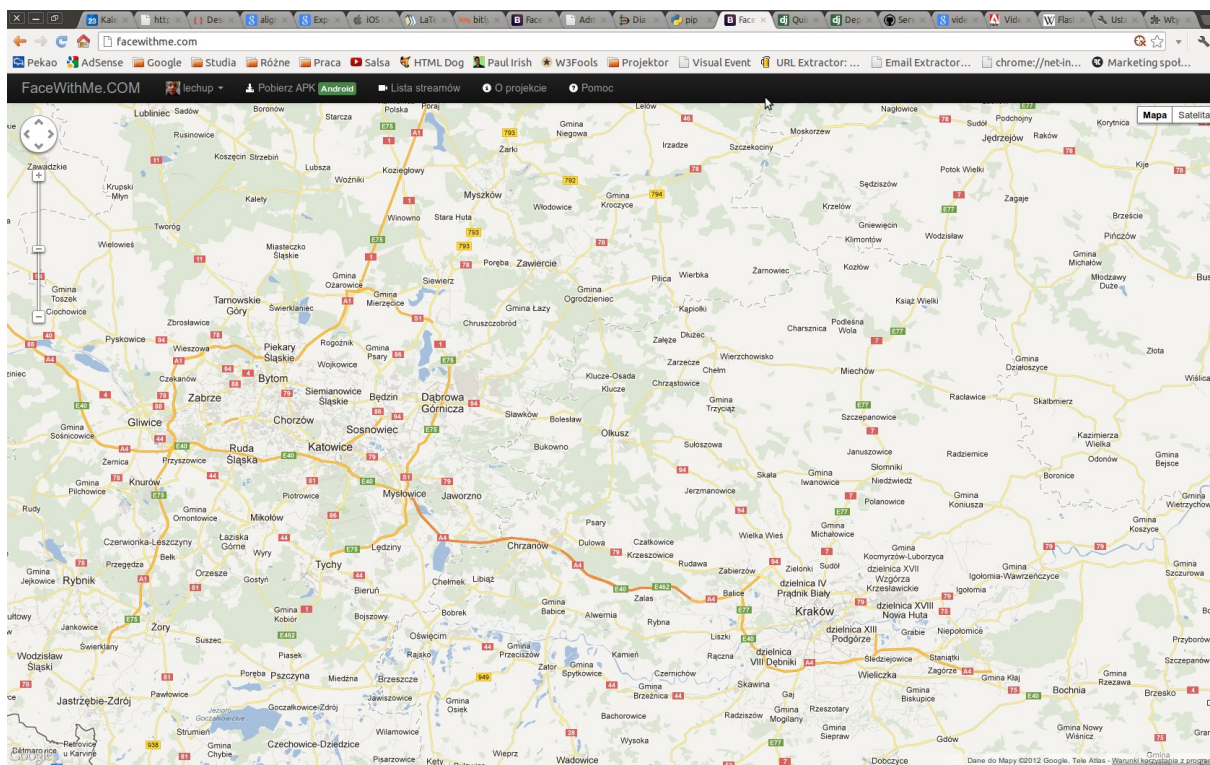
Rysunek 3.2: System rejestracji użytkownika.



Rysunek 3.3: System logowania użytkownika.

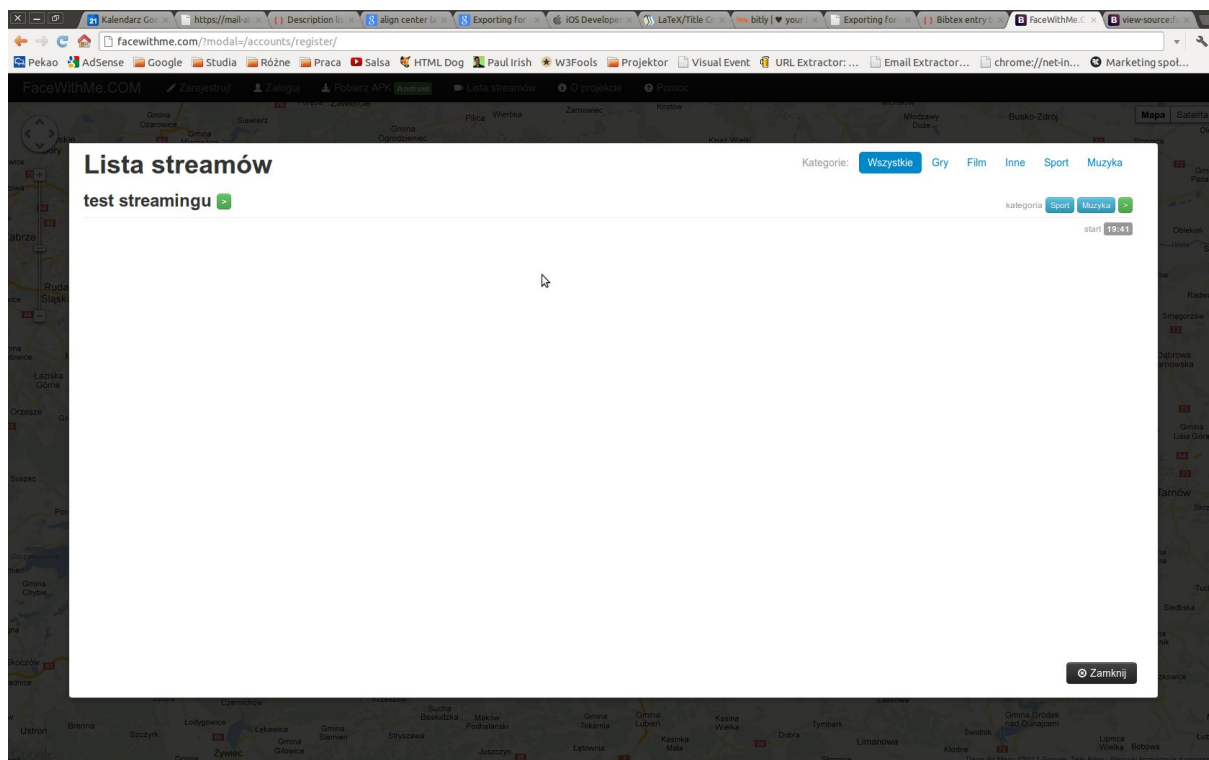


Rysunek 3.4: Panel administracyjny do zarządzania streamami, kategoriami i użytkownikami.

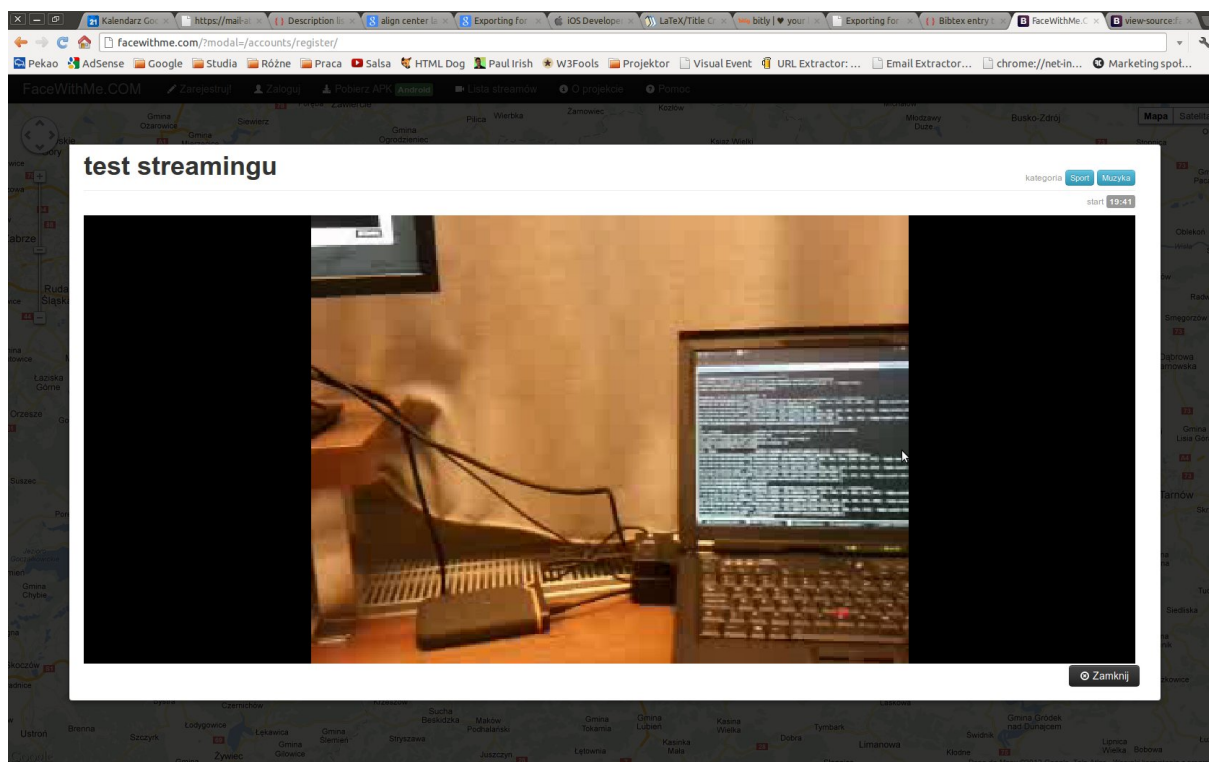


Rysunek 3.5: Interaktywna mapa prezentująca streamy.





Rysunek 3.6: Lista streamów nadawanych w systemie z podziałem na kategorie.

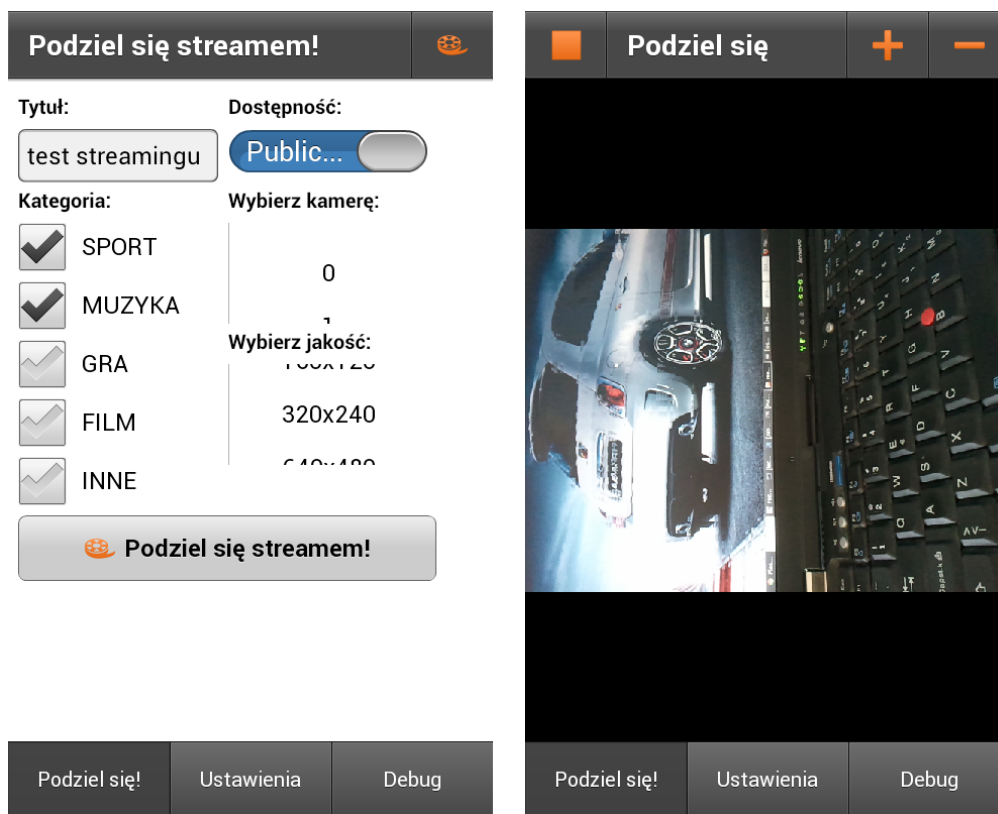


Rysunek 3.7: Wyświetlanie Browser Receivera w Google Chrome.

### 3.1.2. Mobile Broadcaster

Mobile Broadcaster, czyli aplikacja umożliwiająca nadawanie streamu – szczegóły w rozdziale 2.2.2 Wstępna architektura systemu – jest aplikacją na systemy Android wykonana w środowisku AIR. Na przygotowanej stronie <http://facewithme.com> można pobrać aplikację. Aplikacji nie umieszczono w Google Play, gdyż nie jest ona produktem skończonym, a tylko takie można umieszczać na tej platformie. Aplikacja w formie instalatora na systemy Android została umieszczona również na dołączonej płycie CD w katalogu `./src/Django/facewithme/apps/core/static/FaceWithMe.apk`. Aplikacja została przetestowana z wykorzystaniem środowisk uściślonych w rozdziale 3.3.3 Przeprowadzone testy. Niżej przedstawiono funkcje Mobile Broadcastera, które udało się zaimplementować w prototypie systemu.

- Panel udostępniania streamu wideo, można zobaczyć na rysunku 3.8a.
- Streaming wideo, można zobaczyć na rysunku 3.8b.
- Ustawianie jakości streamu w trakcie nadawania, można zobaczyć na rysunku 3.9a.
- Panel ustawień aplikacji, można zobaczyć na rysunku 3.9b.
- Panel DEBUG aplikacji, można zobaczyć na rysunku 3.9c.

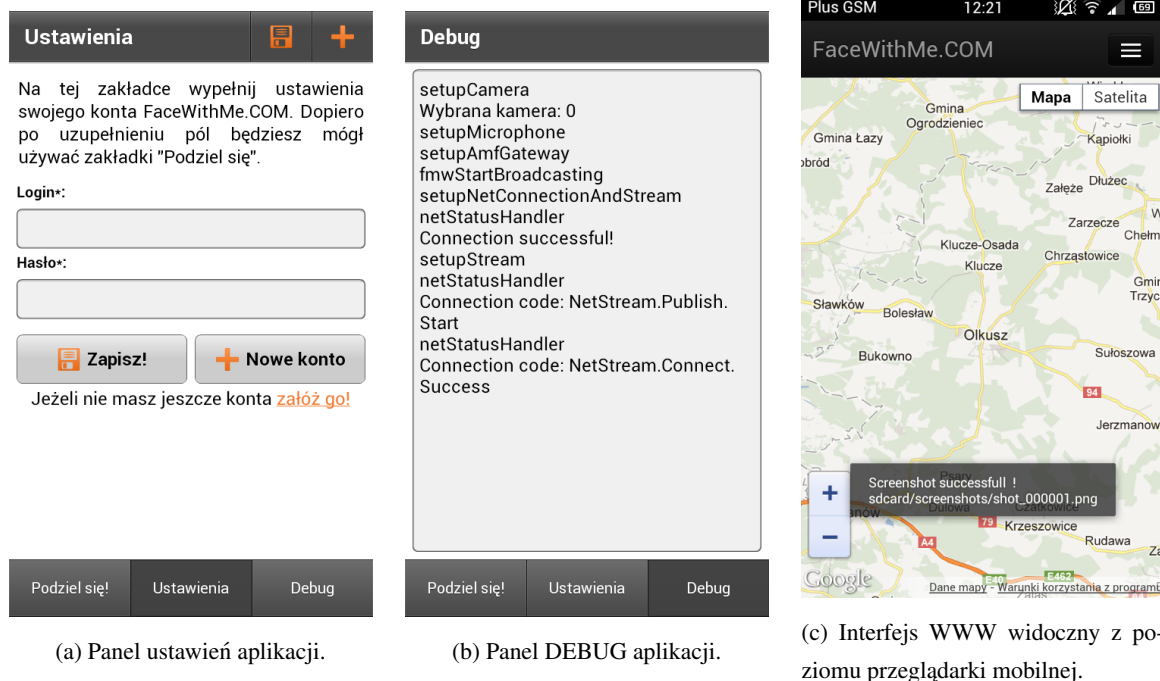


(a) Panel udostępniania streamu.

(b) Streaming wideo.

Rysunek 3.8: Screenshoty z Mobile Receivera część 1.





Rysunek 3.9: Screenshoty z Mobile Receiver'a część 2.

### 3.2. Co wymaga usprawnienia

Przy implementacji prototypu systemu, nie skupiano się na sprawach pobocznych jak interfejs czy komunikacja z użytkownikiem. Głównie kierowano się intencją przetestowania w praktyce działania technologii Adobe P2P Multicast'ing. Dlatego też system wymaga znacznej ilości poprawek zanim może zacząć być wykorzystywany publicznie. Przygotowany prototyp traktować można jako bazę wyjściową do zapoznania się z działaniem i wydajnością technologii. Główne braki prototypu systemu to:

- Brak szyfrowania połączeń HTTP pomiędzy użytkownikiem a systemem, dotyczy także logowania.
- Brak jakiegokolwiek autoryzacji serwera RTMFP, jest dostępny publicznie.
- Brak zabezpieczenia publicznego/prywatnego streamu wideo.
- Brak całego Mobile Receiver'a, szczegóły w rozdziale 2.2.2 Wstępna architektura systemu.
- Brak całego Browser Broadcaster'a, szczegóły w rozdziale 2.2.2 Wstępna architektura systemu.
- Istnienie panelu DEBUG w Mobile Broadcasterze.
- Brak obrotu komponentu kamery względem obrócenia urządzenia w Mobile Broadcasterze co skutkuje przesuwaniem obrazu kamery góra/dół, gdy kamera fizycznie poruszana jest prawo/lewo.
- Brak automatycznego usuwania stream'u z Interfejsu WWW w przypadku nieoczekiwanego przerwania działania Mobile Broadcaster'a.
- Brak komunikatów skierowanych do użytkownika Mobile Broadcaster'a, informujących o stanie lub błędach aplikacji.
- Brak wyłączania trybu czuwania urządzenia podczas udostępniania streamu.
- Brak aplikacji na iOS – Głównym powodem dla którego nie udało się stworzyć i przetestować aplikacji na platformę iOS jest koszt licencji Apple Developer Account (99\$), jaki trzeba ponieść, aby dostać możliwość umieszczania aplikacji na Apps Market oraz testowania aplikacji na własnym telefonie (szczegóły można

znaleźć [Unk11]). Z powodu zastosowania technologii Adobe AIR for Mobile, nie ma możliwości testowania aplikacji za pomocą XCode i wirtualnego środowiska iOS. Z tego powodu dostępność komputerów Mac na uczelni na wiele się nie zdała.

- Brak środowiska testującego oraz testów kodu źródłowego Adobe Flash Platform, zarówno kodu Adobe AIR for Mobile jak i Adobe Flash.

### 3.3. Dokumentacja

Zgodnie z filozofią metodyki „XP” (szczegóły w rozdziale 1.3.1 Założenia), dokumentacja projektu sprowadza się do opisu architektury oraz komentarzy kodu, testów oraz komentarzy do testów. Unika się tworzenia bazy wiedzy, która najczęściej jest po pewnym czasie nieaktualizowana.

Dlatego też, aby zapoznać się z działaniem systemu należy przejrzeć architekturę systemu, a następnie zapoznać się bezpośrednio z kodem źródłowym i zawartymi w nim komentarzami.

#### 3.3.1. Uruchomienie systemu

Aby uruchomić system z kodu źródłowego należy przygotować szereg aplikacji i skonfigurować odpowiednio środowisko. Niżej prezentowana jest lista części systemu, wraz z opisem jak daną część uruchomić.

**Interfejs WWW** Uruchomić aplikację Django, znajdującą się w dołączonym kodzie źródłowym w katalogu `./src/Django/facewithme/`. Wymagane do uruchomienia biblioteki Python’owskiego w formacie pliku wymagań programu PIP znajdują się w `./src/Django/facewithme/requirements/requirements.txt`. Instrukcja instalacji i wdrożenia aplikacji Django znajduje się tutaj: <https://docs.djangoproject.com/en/1.4/>. System wdrożony został przy wykorzystaniu Django 1.4.1

**Baza danych** Uruchomić i zainstalować serwer PostgreSQL. Zainstalować rozszerzenie GIS zgodnie z instrukcją <https://docs.djangoproject.com/en/1.4/ref/contrib/gis/install/#post-installation>. System wdrożony został przy wykorzystaniu PostgreSQL 9.1.

**Serwer RTMFP** Uruchomić serwer Cumulus. Instrukcja instalacji znajduje się na stronie <https://github.com/OpenRTMFP/Cumulus/wiki/Installation>. Dodatkowo należy umieścić serwer na liście dostępnych serwerów w panelu administracyjnym aplikacji Django. Domyślnie, dodawane stream’y losują jeden z dostępnych serwerów RTMFP. Można pokusić się o implementację algorytmu rozkładającego obciążenie bardziej równomiernie np. Round Robin.

**Browser Receiver** Należy tak skonfigurować dołączanie obiektu flash, aby przekazywać informację na temat AMFGateway. Można do tego wykorzystać zmienną `amfgateway` dodając ją w widokach wyświetlających Browser Receiver. Domyślnie ustawiona jest na wartość `http://facewithme.com/amf/`.

**Mobile Broadcaster** Należy uruchomić, zainstalować i skonfigurować środowisko Flash Develop zgodnie z tą instrukcją <http://www.flashdevelop.org/wikidocs/index.php?title=Configuration>. W plikach źródłowych odnaleźć domyślną konfigurację AMFGateway (`./src/Flex/FaceWithMe Mobile/src/Main.mxml`) i poprawić ją na adres skonfigurowanego wcześniej Interfejsu WWW. Skompilować projekt oraz stworzyć odpowiednie paczki instalacyjne dla platform docelowych (Android / iOS).

W razie problemów z wyświetlaniem się streamu należy zapoznać się z plikiem pomocy <http://facewithme.com/help/>.

#### 3.3.2. Testy automatyczne

Część Interfejsu WWW systemu posiada załączki testów jednostkowych. Dotyczą one poprawnego wyświetlania widoków. Aby uruchomić testy, należy przejść do katalogu z z kodem źródłowym Django (`./src/Dajango/facewithme/`), a następnie uruchomić polecenie:

Listing 3.1:

```
1 | python manage.py test core
```

O poprawnym wykonaniu testów informuje nas konsola, której wyjście zawiera:

Listing 3.2:

```
1 | Creating test database for alias 'default'...
   .....
3 |
   Ran 6 tests in 0.822s
5 |
   OK
7 | Destroying test database for alias 'default'...
```

### 3.3.3. Przeprowadzone testy

System testowany był na kilku platformach. Niżej znajduje się lista konfiguracji systemu, w której poszczególne części nie wykazywały błędnego zachowania.

**Interfejs WWW z Browser Receiver** Google Chrome (21.0.1180.89) + Flash Player (11.3.31.232)/Linux; Firefox 15.0.1 + Flash Player 11.2.202.238/Linux; Google Chrome (21.0.1180.89) + Flash Player (11.3.31.232)/Windows 7; Firefox 15 + Flash Player 11.2.202.238/Windows 7.

**Mobile Broadcaster** Adobe AIR 3.4.0.254/Android 4.04/Samsung Galaxy S I, Adobe AIR 3.4.0.254/Android 3.2/Samsung Galaxy Tab 10.1.

### 3.3.4. Bezpieczeństwo

Bardzo ważnym aspektem docelowego systemu powinno być bezpieczeństwo. Niżej wyszczególniono elementy prototypu systemu, które mają bezpośredni związek z bezpieczeństwem i sugerowane poprawki, które pomogą zabezpieczyć system.

- Dzięki zastosowaniu technologii Adobe Flash Platform oraz wykorzystaniu protokołu RTMFP, transmisja danych pomiędzy Mobile Broadcaster'em oraz Browser Receiver'em jest szyfrowana. Szyfrowanie odbywa się za pomocą 128-bitowego klucza. Szczegółowe informacje można uzyskać pod tym adresem [http://help.adobe.com/en\\_US/flashmediaserver/techoverview/WS5b3ccc516d4fbf351e63e3d119ed944af6-7ffb.html](http://help.adobe.com/en_US/flashmediaserver/techoverview/WS5b3ccc516d4fbf351e63e3d119ed944af6-7ffb.html)
- Komunikacja pomiędzy Interfejsem WWW, a Browser Receiver'em oraz Mobile Receiver'em, odbywająca się za pomocą AMFGateway powinna odbywać się za pomocą protokołu HTTPS, nie HTTP jak w prototypie.
- Logowanie do panelu administracyjnego oraz logowanie użytkowników powinno wykorzystywać protokół HTTPS, nie HTTP jak w prototypie.
- Sugeruje się przejście na HTTPS w każdej komunikacji nawiązywanej pomiędzy użytkownikiem, a Interfejsem WWW, nie HTTP jak w prototypie.
- Serwer RTMFP (Cumulus), nie udostępnia żadnego sposobu na autoryzację połączeń. Najprostszym sposobem na ograniczenie wykorzystania przez nieautoryzowanych użytkowników jest sprawdzanie swfUrl łączącego się klienta za pomocą rozszerzenia serwera wykorzystującego zdarzenie onConnection (szczegóły w dokumentacji Cumulus <https://github.com/OpenRTMFP/Cumulus/wiki/Server-Application,-API>)

### 3.4. Wydajność P2P Multicast'ing w Adobe Flash Platform

W tym rozdziale chciano poruszyć temat związany z wydajnością transmisji wideo za pomocą technologii P2P Multicast'ing. Kryterium jakie chcemy sprawdzać to opóźnienie transmisji wideo względem obrazu oryginalnego oraz jego płynność. Opóźnienie jest wyrażone w sekundach. Płynność określono jako jedną z poniższych możliwości.

**Obraz całkowicie płynny** Obraz zachowuje płynność ciągle.

**Obraz płynny** Obraz od czasu do czasu wstrzymuje się, większą część czasu jest płynny.

**Obraz skaczący** Obraz nie jest płynny, posiada natomiast więcej niż jedną klatkę na sekundę.

**Obraz poklatkowy** Obraz nie jest całkowicie płynny, posiada mniej niż jedną klatkę na sekundę.

**Obraz nieczytelny** Obraz pojawia się we fragmentach, wyświetlają się pomniejsze.

Środowisko testowe składa się z dwóch komputerów klientów, na których uruchomiono przeglądarkę Google Chrome i włączono Browser Receiver dla stream'u udostępnianego przez Mobile Broadcaster uruchomionego na Adobe AIR 3.4.0.254/Android 3.2/Samsung Galaxy Tab 10.1. Aplikacja mobilna wysyłanie stream'u wideo przeprowadza za pomocą łącza ADSL (połączenie WiFi przez router) o parametrach faktycznych 8 Mb/s pobieranie, 1 Mb/s wysyłanie danych. Alternatywnym połączeniem Mobile Broadcaster'a jest darmowy internet od Aero2 w technologii HSPA/HSPA+ o parametrach faktycznych 0.5 Mb/s pobieranie, 0.4 Mb/s wysyłanie danych. Komputery klienckie są połączone do tej samej sieci LAN, korzystającej z wcześniej wymienionego łącza ADSL. Test prędkości łącz został wykonany za pomocą serwisu <http://speedtest.net>. W poniższej tabeli znajdują się parametry łącz na których przeprowadzono testy.

	ping	prędkość pobierania	prędkość wysyłania
ADSL	51 ms	8 Mb/s	1 Mb/s
Aero2	107 ms	0.5 Mb/s	0.4 Mb/s

Tablica 3.1: Parametry łącz na których zostały przeprowadzone testy.

Wpływ na badane kryteria mają parametry:

- Rozdzielczość obrazu w transmisji wideo, która jest przekazywana.
- Ilość klientów uczestniczących w transmisji P2P Multicast.
- Łącze wykorzystywane przez Mobile Broadcastera.
- Łącze wykorzystywane przez Browser Receivera.

Tabela przedstawiająca płynność transmisji oglądanej na ekranie Browser Receiver'a, względem rozdzielczości przesyłanego strumienia w pikselach. Dla dwóch włączonych Browser Receiverów, po jednej instancji per komputer klient:

	80x60px	160x120px	320x240px	640x480px
ADSL	obraz całkowicie płynny	obraz całkowicie płynny	obraz płynny	obraz skaczący
Aero2	obraz płynny	obraz poklatkowy	obraz nieczytelny	obraz nieczytelny

Tablica 3.2: Płynność transmisji wideo w danej rozdzielczości względem wykorzystywanego łącza dla dwóch Browser Receiverów.

Tabela przedstawiająca opóźnienie transmisji oglądanej na ekranie Browser Receiver, względem rozdzielczości przesyłanego strumienia w pikselach. Dla dwóch włączonych Browser Receiverów, po jednej instancji per komputer klient:

	80x60px	160x120px	320x240px	640x480px
ADSL	opóźnienie 0s	opóźnienie 0s	opóźnienie 0,1s	opóźnienie 0-1s
Aero2	opóźnienie 0,1-0,5s	opóźnienie 2-3s	opóźnienie 4-5s	opóźnienie 40s

Tablica 3.3: Opóźnienie transmisji wideo w danej rozdzielczości względem wykorzystywanego łącza dla dwóch Browser Receiverów.

Tabela przedstawiająca płynność transmisji oglądanej na ekranie Browser Receiver, względem rozdzielczości przesyłanego strumienia w pikselach. Dla sześciu włączonych Browser Receiverów, po trzy instancje per komputer klient:

	80x60px	160x120px	320x240px	640x480px
ADSL	obraz całkowicie płynny	obraz płynny	obraz płynny	obraz poklatkowy
Aero2	obraz płynny	obraz poklatkowy	obraz poklatkowy	obraz nieczytelny

Tablica 3.4: Płynność transmisji wideo w danej rozdzielczości względem wykorzystywanego łącza dla sześciu Browser Receiverów.

Tabela przedstawiająca opóźnienie transmisji oglądanej na ekranie Browser Receiver, względem rozdzielczości przesyłanego strumienia w pikselach. Dla sześciu włączonych Browser Receiverów, po trzy instancje per komputer klient:

	80x60px	160x120px	320x240px	640x480px
ADSL	opóźnienie 0s	opóźnienie 0-0,3s	opóźnienie 0-0,5s	opóźnienie 0,5-2s
Aero2	opóźnienie 0-3s	opóźnienie 30s	opóźnienie 35s	opóźnienie 45s

Tablica 3.5: Opóźnienie transmisji wideo w danej rozdzielczości względem wykorzystywanego łącza dla sześciu Browser Receiverów.

### 3.5. Wnioski

W tym rozdziale zaprezentowane są wnioski powstałe podczas implementacji prototypu systemu.

- Im gorsze połączenie Mobile Broadcastera tym gorsza możliwa jakość przesyłu wideo.
- Im gorsze połączenie Mobile Broadcastera, a ilość Browser Receiverów większa tym lepsza płynność transmisji wideo.
- Im lepsze połączenie Mobile Broadcastera tym lepsza możliwa jakość przesyłu wideo.
- Im więcej klientów Browser Receiverów podłączone do stream'u tym większe możliwe opóźnienie.
- Aby stworzyć aplikację na iOS za pomocą Adobe AIR for Mobile należy posiadać Apple Developer Account.
- Adobe P2P Multicasting działa i wydaje się sensowną technologią do wdrożenia projektowanego systemu.
- Aby publicznie udostępnić system należy sporo rzeczy poprawić, w szczególności popracować nad bezpieczeństwem systemu.
- Komunikacja P2P odbywa się przy udziale wysokiego portu UDP. Komunikacja ta może być blokowana w sieciach korporacyjnych.
- Wszystkie wykorzystane przy implementacji prototypu technologie spełniają swoją powierzoną rolę.
- Istnieje ryzyko stworzenia systemu służącego do permanentnej inwigilacji...
- Metodyka XP ma duże szanse powodzenia przy prowadzeniu szybko zmieniającego się projektu przy udziale niedużej grupy projektowej.
- Przygotowywanie ekranów do kart wymagań za pomocą programu graficznego jest pracochłonną czynnością, dużo lepiej powinny sprawdzić odręczne szkice i „cyfryzacja” za pomocą zdjęć np. telefonem komórkowym.

## A. L<sup>A</sup>T<sub>E</sub>X, środowisko `userstory`

Na potrzeby niniejszej pracy autor musiał wymyślić sposób na cyfryzację kart wymagań, związanych z nimi testów akceptacyjnych oraz ekranów, tak aby w łatwy sposób dało się je dołączyć do pracy. Tak powstał tzw. „package” udostępniający środowisko `userstory`. Więcej o dodatkowych zaletach cyfryzacji można przeczytać w rozdziale 1.3.6 Cyfryzacja i zarządzanie.

Należy pamiętać, że każda metodyka zwinna jest przeciwna generowaniu bezużytecznej dokumentacji. Należy więc wziąć to pod uwagę przy wykorzystywaniu środowiska „`userstory`”.

### A.1. Sposób użycia

Aby skorzystać ze środowiska należy wykonać dwie rzeczy:

- do katalogu dokumentu L<sup>A</sup>T<sub>E</sub>X wgrać plik `userstory.sty`
- do generowanego dokumentu dołączyć nagłówek:

Listing A.1:

```
1 | \usepackage{userstory}
```

Po takim zabiegu w kodzie dostępne jest środowisko `userstory`. W jego wnętrzu zapisujemy treść karty wymagań, dodatkowo mamy do dyspozycji:

- Środowisko `tests` do definiowania nowych testów akceptacyjnych. Za pomocą komendy `item` możemy zdefiniować w nim kolejny test akceptacyjny.
- Środowisko `questions` do definiowania pytań do klienta, które pojawiły się w „międzyczasie”. Za pomocą komendy `item` możemy zdefiniować w nim kolejne pytanie.
- Dwuargumentową komendę `src$file$caption` do umieszczania ekranów w dowolnym miejscu środowiska (przy pytaniach, przy samej karcie wymagań albo przy konkretnym teście akceptacyjnym). Argument `$file` to ścieżka do pliku graficznego, a `$caption` to opis który będzie dodany pod ekranem.

### A.2. Przykład użycia

Poniżej przygotowany przykład wykorzystania środowiska `userstory` celem przygotowania karty wymagań „Logowania użytkownika”.



Listing A.2: userstory–latex/logowanie–uzytkownika.tex

```

1 % domyślne formatowania dokumentu, można zmienić
  \documentclass[a4paper]{article}
3 \usepackage{fullpage}
  \usepackage[utf8]{inputenc}
5 \usepackage{polski}
  \usepackage[polish]{babel}
7
  % dodanie wymaganego nagłówka środowiska userstory
9 \usepackage{userstory}
11
  \begin{document}
    \begin{userstory}{Logowanie użytkownika}
13      Użytkownik za pomocą formularza może zalogować się do serwisu,
        uzyskując w ten sposób dostęp do jego dodatkowych funkcjonalności.
15      \scr{img/us1/1.png}{Formularz logowania.}

17      \begin{tests}
        \item{
19          Użytkownik tylko po podaniu podaniu loginu oraz pasującego do niego hasła
            zostaje zalogowany i przeniesiony do pulpitu.
21        }
        \item{
23          Użytkownik po podaniu błędnego hasła lub nieistniejącego loginu,\\*
            zostaje o tym poinformowany oraz oferuje mu się od razu formularz
            przypomnienia hasła.
25          \scr{img/us1/2.jpg}{Ekran informacji o błędnym hasle lub loginie.}
        }
27        \item{
          Użytkownik po zalogowaniu\\*
29          widzi zmodyfikowane menu na wszystkich stronach serwisu.
          \scr{img/us1/2.jpg}{Zakładki menu dla zalogowanego użytkownika.}
31          \scr{img/us1/3.jpg}{Zakładki menu dla niezalogowanego użytkownika.}
        }
33        \item{
          Użytkownik po zalogowaniu\\*
35          ma dostęp do panelu.
        }
37        \item{
          Użytkownik po zalogowaniu\\*
39          może przeglądać profile innych użytkowników.
        }
41      \end{tests}
    \begin{questions}
43      \item{
        Czy użytkownik po wykonaniu błędnego logowania\\*
45        ma być przekierowany na nową stronę, czy pozostawać na tej samej?
      }
47    \end{questions}
  \end{userstory}
49 \end{document}

```

## A.3. Kod środowiska

Poniżej znajduje się kod wykorzystywany w celu definicji środowiska.

Listing A.3: userstory–latex/userstory.sty

```

1 \ProvidesPackage{userstory}
3 % requirements
\NeedsTeXFormat{LaTeX2e}
5 \RequirePackage{graphicx}
\RequirePackage{capt-of}
7
\newenvironment{userstory}[1]{
9   \newcommand{\scr}[2]{
      \begin{center}
11         \includegraphics[width=0.80\textwidth]{##1}
      \captionof{figure}{##2}
13     \end{center}
    }
15   \newenvironment{tests}{
      \subsubsection[Testy akceptacyjne]{Testy akceptacyjne\footnote{Testy powinny być
        pisane \emph{przez klienta}, jedynie \emph{pod nadzorem} programisty!}}
17     \begin{itemize}
        \let\itemi\item
19         \renewcommand{\item}[1]{
            \begin{itemi}
21               \textit{\noindent\ignorespaces ####1}
            \end{itemi}
23         }
      }
25   {
      \let\item\itemi
27     \end{itemize}
    }
29   \newenvironment{questions}{
      \subsubsection[Pytania do klienta]{Pytania do klienta\footnote{Pytania powinny pojawia
        ć się od razu przy rozmowie z klientem tak aby dało się w rozmowie wyjaśnić o co
        dokładnie chodzi. Każde z pytań niedotyczące głównej karty wymagań systemu powinno
        wygenerować test akceptacyjny lub zmodyfikować istniejący!}}
31     \begin{itemize}
        \let\itemi\item
33         \renewcommand{\item}[1]{
            \begin{itemi}
35               \textit{\noindent\ignorespaces ####1}
            \end{itemi}
37         }
      }
39   {
      \let\item\itemi
41     \end{itemize}
    }
43
\subsection{Karta wymagań ,,\emph{#1}''}
45 \begin{textit}
    \noindent\ignorespaces
47 }
```

```

49 {
    \end{textit}
    \newpage
51 }

```

## A.4. Szablon „Główne zadanie systemu”

Poniżej znajduje się kod odpowiedzialny za wygenerowanie szablonu głównego zadania systemu.

Listing A.4: userstory-latex/szablon-glowne-zadanie-systemu.tex

```

1 \documentclass[a4paper]{article}
  \usepackage{fullpage}
3 \usepackage[utf8]{inputenc}
  \usepackage{polski}
5 \usepackage[polish]{babel}

7 \usepackage{userstory}

9 \begin{document}
  \begin{userstory}{Główne zadanie systemu}
11   Tutaj opisać ogólnie główne zadanie systemu, bez realizacji którego nie będzie można
      powiedzieć, że system działa. Jak system składa się z kilku podsystemów, dla każ-
      dego systemu przygotować podobną kartę wymagań.
  \begin{questions}
13     \item{
        \textbf{Kto jest użytkownikiem systemu?} Odpowiedź
15     }
      \item{
17       \textbf{Jakie urządzenia muszą współpracować z systemem?} Odpowiedź
      }
19     \item{
        \textbf{Czy jest wymóg użycia konkretnej technologii?} Odpowiedź
21     }
      \item{
23       \textbf{Jaka jest wymagana skalowalność systemu?} Odpowiedź
      }
25     \item{
        \textbf{Czy są jakieś wymagania dotyczące środowiska produkcyjnego w jakim ma
          działać system?} Odpowiedź
27     }
      \item{
29       \textbf{Czy system ma współpracować z innymi systemami lub udostępniać coś
          innym systemom?} Odpowiedź
      }
31     \item{
        \textbf{Czy istnieją systemy podobne do tworzonego?} Odpowiedź
33     }
      \item{
35       \textbf{Czy są jakieś inne specjalne wymagania, które nie wynikają z funkcji
          jakie powinien posiadać system (wymagania нефункциональные)?} Odpowiedź
      }
37   \end{questions}
  \end{userstory}
39 \end{document}

```

## **B. Podziękowania autora**

Serdeczne podziękowania należą się promotorowi dr inż. Sebastianowi Ernstowi, za trzymanie pracy w ryzach, cenne wskazówki oraz ogromny dystans i pozytywną energię podczas spotkań poświęconych niniejszej pracy magisterskiej.

Praca nie powstałaby, gdyby nie szereg osób motywujących do działania. Szczególne podziękowania należą się Genowefie Dobrowolskiej, znanej bardziej autorowi pracy jako „babcia Genia”, Jadwidze Piątek do której autor na co dzień zwraca się per „mama” oraz Aleksandrze Piątek – kochanej siostrze.

Podziękowania od autora należą się również współpracownikom z firmy Quercus oraz Agencja Bracia Sadurscy, za cierpliwość i wyrozumiałość dla niedostępności wynikającej z całkowitego poświęcenia się pracy magisterskiej.

# Spis rysunków

2.1	Strona główna dla niezalogowanego użytkownika . . . . .	17
2.2	Strona główna dla zalogowanego użytkownika . . . . .	19
2.3	Formularz rejestracji. . . . .	20
2.4	Formularz przypomnienia hasła. . . . .	21
2.5	Ekran informacji o błędnym wypełnieniu loginu/hasła. . . . .	22
2.6	Formularz logowania. . . . .	23
2.7	Ekran informacji o błędnym hasle lub loginie. . . . .	24
2.8	Zmodyfikowane menu na górze . . . . .	25
2.9	Formularz udostępniania audio/video. . . . .	26
2.10	Okno danych udostępnianego streamu audio/video. . . . .	28
2.11	Przedstawienie transmisji Unicast w Adobe Flash Platform, obrazek pochodzi z [Krc10]. . . . .	30
2.12	Przedstawienie transmisji P2P Multicast w Adobe Flash Platform, obrazek pochodzi z [Krc10]. . . . .	31
2.13	Wstępna architektura systemu. . . . .	32
2.14	Logo Adobe Flash Platform, pobrane z Google Images . . . . .	33
2.15	Logo Django, pobranie ze strony projektu. . . . .	36
2.16	Logo jQuery, pobranie ze strony projektu. . . . .	37
2.17	Logo Bootstrap, pobranie ze strony projektu. . . . .	38
2.18	Logo FlashDevelop, pobranie ze strony projektu. . . . .	39
2.19	Logo PostgreSQL, pobranie ze strony projektu. . . . .	40
2.20	Logo Google Developers, odpowiedzialnego za Google Maps JavaScript API, pobranie ze strony projektu. . . . .	41
3.1	Architektura implementacji prototypu systemu uwzględniająca transmisję wideo z Mobile Broad-castera do kilku Browser Receiverów. . . . .	44
3.2	System rejestracji użytkownika. . . . .	45
3.3	System logowania użytkownika. . . . .	45
3.4	Panel administracyjny do zarządzania streamami, kategoriami i użytkownikami. . . . .	46
3.5	Interaktywna mapa prezentująca streamy. . . . .	46
3.6	Lista streamów nadawanych w systemie z podziałem na kategorie. . . . .	47
3.7	Wyświetlanie Browser Receivera w Google Chrome. . . . .	47
3.8	Screenshoty z Mobile Receiver'a część 1. . . . .	48
3.9	Screenshoty z Mobile Receiver'a część 2. . . . .	49

## Bibliografia

- [Bec99] K. Beck. Extreme programming explained, 1999.
- [Jef00] R. Jeffries. Extreme programming installed, 2000.
- [JM06] M. Jakała and M. Michno. Projekt i implementacja internetowego systemu obsługi konferencji, 2006. <http://bit.ly/JakMich06>.
- [Kau09] M. Kauffman. P2P on the Flash Platform with RTMFP, 2009. <http://bit.ly/MattKauf2009>.
- [Kim11] R. Kim. Mobile internet user to eclipse wireline users by 2015, 2011. <http://bit.ly/Kim11>.
- [KKP00] R. Kazman, M. Klein, and Clements P. ATAM: Method for Architecture Evaluation, 2000. <http://bit.ly/Kaz2000>.
- [Krc10] T. Krcha. Multicast Explained in Flash 10.1 P2P, 2010. <http://bit.ly/TomKrcha2010ME>.
- [MK09] L. Madeyski and M. Kubiasiak. Zwinna specyfikacja wymagań, 2009. <http://bit.ly/Mad09>.
- [Unk11] A. Unknown. Exporting for iPhone using AIR 2.7 and FlashDevelop, 2011. <http://bit.ly/UnknAuth11>.