Akademia Górniczo-Hutnicza

im. Stanisława Staszica w Krakowie

Wydział Informatyki, Elektroniki i Telekomunikacji

Alarm

Opis testów systemu

Mateusz Gaweł

Jędrzej Dąbrowa

Mateusz Konieczny

Przemysław Kurc

Wykonał: Jędrzej Dąbrowa

Alarm - plan testów systemu

Jako część dobrej praktyki inżynierskiej, w trakcie prac nad implementacją systemu Alarm szczególny nacisk położony zostanie na wyposażenie projektu w odpowiedni zestaw testów. Testy te odbywać się będą na kilku poziomach, z osobnymi kryteriami odnośnie każdego z nich. Podział testów na grupy przedstawia się następująco:

1. Testy aplikacji klienckiej
    a. Testy jednostkowe poszczególnych klas i komponentów
    b. Testy integracyjne aplikacji
    c. Testy manualne aplikacji

2. Testy serwera
    a. Testy jednostkowe poszczególnych klas i komponentów
    b. Testy integracyjne aplikacji
    c. Testy automatyczne REST API z użyciem odpowiedniego narzędzia

3. Testy integracyjne z użyciem obu komponentów
    a. Testy manualne z wykorzystaniem wielu urządzeń

Poniżej przedstawiono plan szczegółowej realizacji testów na poszczególnych etapach:

1. Testy aplikacji klienckiej

b. Testy jednostkowe poszczególnych klas i komponentów
Każda klasa (lub inna jednostka funkcjonalności, zależnie od kontekstu i konkretnego rozwiązania) powinna posiadać odpowiedni zestaw automatycznych testów, uruchamianych w ramach budowy projektu. Projekt oparty jest o system *gradle*, i przy jego użyciu uruchamianie wszystkich testów jednostkowych zostanie integralną częścią procesu budowu projektu. Wykorzystane zostaną frameworki Roboelectric, JUnit i Mockito, aby zapewnić narzędzia do tworzenia testów oraz środowisko do szybkiego ich uruchamiania.

c. Testy integracyjne aplikacji

Ten zestaw testów wykorzystuje wiele komponentów na raz i weryfikuje ich poprawne współdziałanie. Wybór narzędzia zostanie dokonany po stworzeniu szczegółowej listy scenariuszy testowych. Dostępne narzędzia to między innymi Espresso, Google Android Testing czy Robotium.

d. Testy manualne aplikacji

Testy wykonywane ręcznie po zakończeniu prac nad modułem aplikacji. Ich podstawowym celem jest weryfikacja płynności i stabilności działania aplikacji oraz poprawnego wyglądu graficznego interfejsu użytkownika.

Testy automatyczne zostaną dodatkowo wzbogacone o informacje z analizy JaCoCo, w celu uzyskania szczegółowych danych o procentowym pokryciu kodu testami oraz identyfikacji miejsc, gdzie zachodzi potrzeba wzbogacenia zestawu testów.

2. Testy serwera

e. Testy jednostkowe poszczególnych klas i komponentów
Podobnie jak w przypadku aplikacji klienckiej, kod serwera zostanie wyposażony w zestaw testów jednostkowych. Testy te realizowane będą z użyciem frameworków JUnit oraz Mockito/EasyMock, a ich uruchomienie odbywać się będzie w ramach buildu narzędzia Maven, zatem każdorazowe budowanie projektu da informacje o przechodzących/nieprzechodzących testach, zapewniając szybką informację po wprowadzeniu jakichkolwiek zmian w kodzie serwera.

f. Testy integracyjne aplikacji

Testy współpracy między komponentami. Jako, że aplikacja serwera oparta zostanie o framework Spring Boot, również część testów zostanie zrealizowana z wykorzystaniem funkcjonalności udostępnianych przez to środowisko. Największa zaleta tego rodzaju testów to między innymi inicjalizacja całości aplikacji (zapewniająca możliwość weryfikacji m.in. poprawności konfiguracji i zależności w grafie obiektów) oraz możliwość testowania wysokopoziomowych przepływów danych i sterowania w całości systemu.

g. Testy automatyczne REST API z użyciem odpowiedniego narzędzia

Testowanie serwera aplikacji jedynie poprzez REST API, z użyciem zewnętrznego narzędzia oraz w oparciu o zdefiniowane scenariusze. Dostępne narzędzia to m.in. Apache JMeter oraz TestNG. Weryfikacja oparta będzie o spodziewane przepływy danych oraz treść zapytań wysyłanych do i odpowiedzi odebranych z serwera.

Podobnie, jak w przypadku testów aplikacji, zostanie wykorzystane narzędzie (JaCoCo) do analizy pokrycia kodu testami. Zastosowane zostanie również środowisko *Continous Integration* (Jenkins), zapewniające automatyzację oraz regularność wykonania testów.

3. Testy integracyjne z użyciem obu komponentów

h. Testy manualne z wykorzystaniem wielu urządzeń

Ostatnim etapem testów będą testy całości systemu, wykorzystujące kilka (różnych) współpracujących urządzeń z systemem Android, łączących się z aktywnym serwerem. Odgrywane będą scenariusze symulujące użycie przez rzeczywistych użytkowników. Weryfikacja obejmować będzie analizę zachowania aplikacji w trakcie testów, poprawności przesyłanych danych, logów po stronie serwera oraz logów aplikacji (z użyciem opcji debugowania udostępnianych przez Android SDK)