Opis

Projekt ten jest implementacją mechanizmu server-push z wykorzystaniem techniki long-polling, przeznaczoną dla Frameworka Spring. Technika ta przydaje się w sytuacji, gdy aplikacja kliencka chce pozyskać nowe dane od serwera - wówczas nie trzeba powtarzać w kółko zapytań otrzymując puste odpowiedzi, lecz serwer może utrzymywać zapytanie otwarte aż do pojawienia się nowych danych i dopiero wtedy odesłać odpowiedź.

Zalety takiego podejścia to:

- Zmniejszenie liczby niepotrzebnych zapytań do serwera
- Wysoka responsywność aplikacji klienckiej (aplikacja kliencka otrzymuje dane natychmiast po ich pojawieniu się)
- Bardzo mała ilość zmian wymagana w aplikacji klienckiej (przy przejściu z “standardowego” pollingu do long-pollingu)

Podejście to przejawia wady takie jak:

- Zwiększenie obciążenia serwera - każde otwarte połączenie będzie zwykle absorbowało jeden wątek serwera (zazwyczaj większość połączeń będzie w bezczynnym stanie oczekiwania)

Opis implementacji

Główny kontroler aplikacji posiada kolejkę obiektów typu “Promise” - reprezentują one początkowo nieznany rezultat jakiegoś działania, który zostanie ustalony w przyszłości. Wraz z każdym zapytaniem typu “long-

polling” do owej kolejki dodawany jest nowy obiekt typu “Promise” reprezentujący to konkretne zapytanie. Zapytanie pozostaje otwarte aż do czasu kiedy wartość zostanie ustalona w tymże obiekcie - kiedy to nastąpi, ustalona wartość zostanie wysłana w ciele odpowiedzi.

Próba ustalenia rezultatu działania w obiektach typu “Promise” następuje cyklicznie poprzez wykonanie metody execute() na każdym z obiektów kolejki. Jeżeli ustalenie rezultatu dla obiektu nastąpiło, wówczas obiekt jest usuwany z kolejki.

Opis aplikacji

Aby zilustrować działanie long-pollingu postanowiliśmy stworzyć aplikację do wysyłania powiadomień. Po otwarciu aplikacji, wszystkie powiadomienia są ładowane z bazy danych. Następnie, aplikacja wysyła zapytanie typu “long-polling”, aby pozyskać nowe powiadomienia. Jeżeli nowe powiadomienie pojawi się, wówczas jego szczegóły zostaną zwrócone w odpowiedzi i aplikacja doda nową notyfikację do widoku aplikacji, po czym otworzy nowe zapytanie typu “long-polling”, aby pozyskać kolejne powiadomienia.

Przykład użycia

Aby użyć stworzonej przez nas biblioteki, kod kliencki musi wykonać następujące kroki:

1. Kontroler aplikacji musi dziedziczyć po MainController

```
@RestController
@RequestMapping("/api")
public class AppController extends MainController {
    ...
}
```

2. Należy zaimplementować własną klasę typu Resolver, której zadaniem jest ustalenie wyniku działania dla obiektów typu “Promise”

```
/**
 * Resolver that is successfully resolving Promises, when new
 record has been added to notification table
 */
@Component
public class NewNotificationResolver implements Resolver, Obs
erver {
    ...
}
```

3. Wraz z żądaniem typu “long-polling”, do kolejki kontrolera musi zostać dodany nowy obiekt typu “Promise” reprezentujący żądanie

```
@RequestMapping(value = "/newNotification", method = RequestM
ethod.GET)
public @ResponseBody
DeferredJSON deferredResult() {
```

```
    DeferredJSON result = new DeferredJSON(resolver);
    supervisor.add(result);
    return result;
}
```

Użyte wzorce projektowe:

1. Data Access Object Pattern

   Poniższe klasy są używane w celu wykonywania operacji na źródle danych (np. bazie danych)

   - AppUserDao
   - NotificationDao

2. Observer Pattern

   - NotificationService jest typu Observable - gdy pojawia się nowa notyfikacja, powiadamia obserwatorów
   - NewNotificationResolver jest typu Observator - “rozwiązuje” zapytanie asynchroniczne po pojawieniu się nowej notyfikacji (wartości w obiekcie typu Promise)

3. Command

   Poniższa klasa realizuje wzorzec projektowy “Command” gdyż enkapsuluje akcję, która jest potrzebna do “rozwiązania” zapytania asynchronicznego

    - DeferredJSON

4. Singleton

   Poniższa klasa realizuje wzorzec Singleton, gdyż jest potrzebna jedynie jedna instancja połączenia z bazą danych

    - DbConnection

5. Promise

   Poniższa klasa realizuje wzorzec “Promise”, gdyż reprezentuje początkowo nieznany wynik jakiegoś działania, który może zostać ustalony w przyszłości

    - DeferredJSON

Uruchamianie aplikacji:

1. W pierwszym kroku należy zaimportować zależności:

```
mvn clean install -U
```

2. Następnie należy wykonać poniższe polecenie:

```
mvn exec:java -Dexec.mainClass="pl.edu.agh.kis.Main" --sp
ring.config.location=classpath:/application.properties
```

gdzie `classpath:/application.properties` oznacza ścieżkę do pliku

konfiguracyjnego aplikacji

Konfiguracja

Konfiguracja aplikacji znajduje się w pliku `src/main/resources/application.properties`

Build

Plik wykonywalny .jar ma nazwę `gs-spring-boot-0.1.0.jar`

Dokumentacja

1. Javadoc - znajduje się w katalogu javadoc/

2. Swagger - znajduje się w pliku ApiDocumentation.pdf lub pod endpointem `/swagger-ui.html` po uruchomieniu aplikacji

Diagramy

Diagram klas (zbyt duży by zmieścić w readme)

Diagram klas

Diagram przypadków użycia

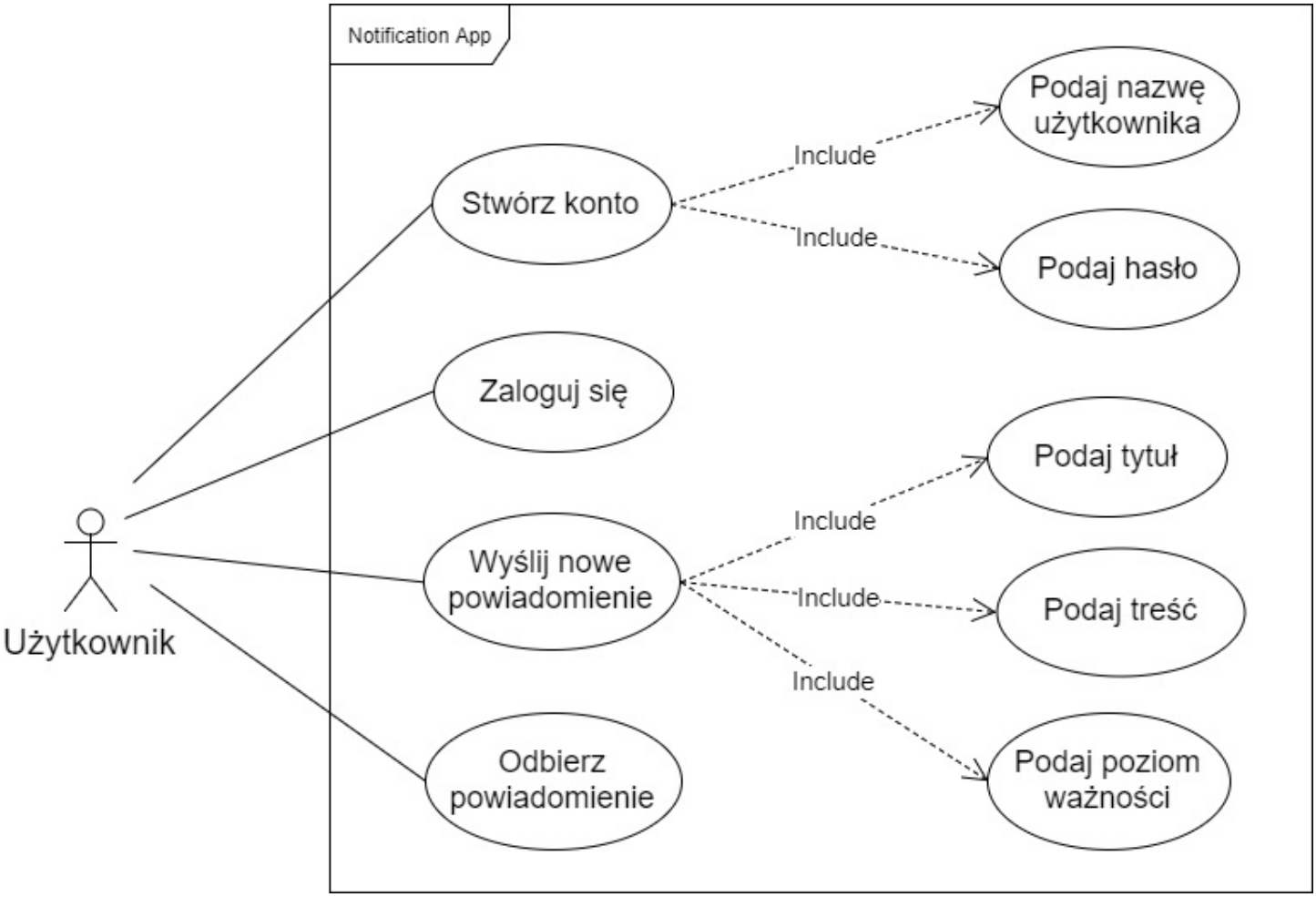
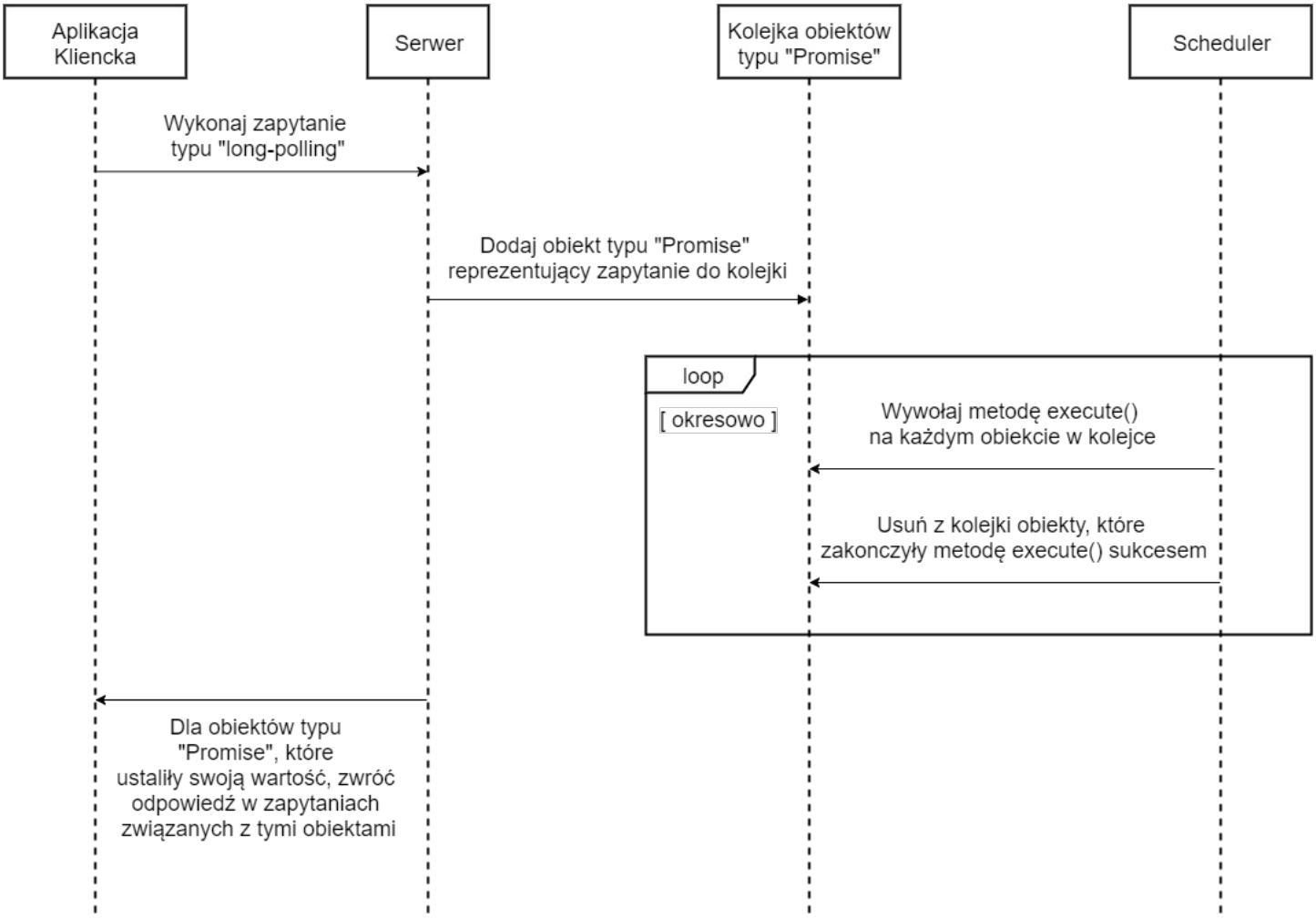


Diagram sekwencji



Screeny aplikacji

Log In

Username

Password

Login

Go To Register

Notifications App Home a

Notifications App

Title

Title of notification

Content

vitae, lacinia dolor. Suspendisse faucibus lacus odio, a laoreet felis ullamcorper at. Aliquam sed auctor magna, quis tincidunt quam. Mauris et neque risus. Praesent non imperdiet elit. Mauris rutrum mattis nulla, in accumsan turpis posuere eu. Morbi eu enim non mi blandit luctus in eget turpis. Vestibulum ac ipsum rhoncus, vestibulum nulla ac, tristique dolor.

Importance level

3

Submit