**AKADEMIA GÓRNICZO-HUTNICZA**

**IM. STANISŁAWA STASZICA W KRAKOWIE**



**Wydział Informatyki, Elektroniki i Telekomunikacji**

Katedra Informatyki

**Dokumentacja Techniczna**

**Interaktywny system wspomagania układania rozkładu zajęć**

Roger Barlik, Krzysztof Nowakowski,

Krystian Ujma, Krzysztof Wróbel,

Promotor: Mgr inż. Witold Rakoczy

Spis treści

[1. Wstęp 4](#_Toc424253296)

[2. Obiekty 4](#_Toc424253297)

[2.1. Agregat 4](#_Toc424253298)

[2.2. Autoryzacja 5](#_Toc424253299)

[2.3. Typ autoryzacji 5](#_Toc424253300)

[2.4. Awatar 6](#_Toc424253301)

[2.5. Sala 6](#_Toc424253302)

[2.6. Salo-godzina 7](#_Toc424253303)

[2.7. Konkretna data 7](#_Toc424253304)

[2.8. Szablon 8](#_Toc424253305)

[2.9. Konkretna Lekcja 8](#_Toc424253306)

[2.10. Wydział 9](#_Toc424253307)

[2.11. Kierunek 9](#_Toc424253308)

[2.12. Oznaczenie grupy dziekanatowej 9](#_Toc424253309)

[2.13. Jednostka programowa 10](#_Toc424253310)

[2.14. Typ jednostki programowej 10](#_Toc424253311)

[2.15. Szablon planu lekcji 11](#_Toc424253312)

[2.16. Grupa studencka 11](#_Toc424253313)

[2.17. Wykładowca 12](#_Toc424253314)

[2.18. Semestr 12](#_Toc424253315)

[2.19. Użytkownik 12](#_Toc424253316)

[4. Architektura 14](#_Toc424253317)

[4.1. Wprowadzenie 14](#_Toc424253318)

[4.2. Komunikacja wysokopoziomowa 14](#_Toc424253319)

[4.3. HTML 15](#_Toc424253320)

[4.4. Spring 16](#_Toc424253321)

[4.5. Moduł Hibernate 17](#_Toc424253322)

[4.5.1. Baza Danych 17](#_Toc424253323)

[4. Diagramy przypadków użycia 19](#_Toc424253324)

[4.1. Zarządzanie planami zajęć 19](#_Toc424253325)

[4.4 Tworzenie zajęcia wraz z terminem 21](#_Toc424253326)

[4.6 Zarządzanie widokami 22](#_Toc424253327)

[4.8 Zarządzanie awatarami 24](#_Toc424253328)

[5. Instalacja 26](#_Toc424253329)

[5.1. Sposób I 26](#_Toc424253330)

[5.2. Sposób II 27](#_Toc424253331)

[6. Materiały źródłowe 28](#_Toc424253332)

[7. Spis ilustracji 28](#_Toc424253333)

# Wstęp

Poniższy tekst jest dokumentacją techniczną opisującą szczegóły architektoniczne i implementacyjne systemu stworzonego w ramach pracy inżynierskiej.

Tematem projektu jest **stworzenie interaktywnego systemu wspomagania układania planu lekcji**. System jest przeznaczony do uruchomiania na serwerze w sieci. System pozwala na tworzenie, przechowywanie, przeglądanie oraz edytowanie planów zajęć.

# Obiekty

W celu wymiany danych pomiędzy modułami i ich zapisu w bazie danych zostały utworzone odpowiednie obiekty. Każdy obiekt posiada swój odpowiednik w każdym z modułów: prototyp w module HTML, klasę w module Hibernate, tabele w bazie danych. W tym rozdziale zostaną zaprezentowane poszczególne obiekty w języku Java.

## Agregat

Agregat jest przedstawieniem połączenia wszystkich danych tworzących przesuwalny prostokąt umieszczany na kalendarzu.

**Definicja:**

public class Aggregate {

private Long id;

private Avatar avatar;

private ProgrammeUnit programmeUnit;

private StudentGroup studentGroup;

}

**id:** unikalny klucz agregatu

**avatar:** awatar przypisany do danego agregatu

**programmeUnit:** - jednostka programowa przypisana do danego agregatu

**studentGroup:** grupa studencka przypisana do danego agregatu

## Autoryzacja

Obiekt przygotowany do przechowywania danych autoryzacyjnych.

**Definicja:**

public class Authorization {

private Long id;

private AuthorizationType type;

private User user;

}

**id:** unikalny klucz autoryzacji

**type:** rodzaj dostępu do którego przypisany jest użytkownik

**user:** nazwa użytkownika

## Typ autoryzacji

Obiekt przechowujący rodzaj dostępu dostępny dla użytkownika

**Definicja:**

public class AuthorizationType {

private Long id;

private String name;

}

**id:** unikalny klucz typu autoryzacji

**name:** rodzaj użytkownika

## Awatar

Awatar przedstawia połączenie kilku wykładowców, którzy wspólnie prowadzą daną jednostkę podstawy programowej (przedmiot) dla danej grupy studenckiej.

**Definicja:**

public class Avatar {

private Long id;

private String name;

private Set<Aggregate> aggregates;

private Set<Teacher> teachers;

}

**id:** unikalny klucz awatara

**name:** unikalna nazwa awatara

**aggregates:** agregaty do których przypisany jest dany awatar

**teachers:** wykładowcy przypisani do danego awataru

## Sala

Obiekt sala jak sama nazwa wskazuje przechowuje informacje na temat danej Sali.

**Definicja:**

public class Classroom{

private Long id;

private String number;

private int buildingId;

private int numberOfSeats;

private Set<ClassroomHour> classroomHours;

}

**id:** unikalny klucz sali

**number:** numer sali

**buildingId:** numer budynku w którym znajduje się sala

**numberOfSeats:** ilość miejsc dostępnych w Sali

## Salo-godzina

Obiekt jest połączeniem Sali oraz czasu w którym odbywa się lekcja.

**Definicja:**

public class ClassroomHour{

private Long id;

private int lessonBeginId;

private int lessonEndId;

private Classroom classroom;

}

**id:** unikalny klucz salo-sodziny

**lessonBeginId**: id bloku w którym zaczyna się lekcja

**lessonEndId**: id bloku w którym kończy się lekcja

**classroom:** przypisana sala

## Konkretna data

Konkretna data odzwierciedla konkretny punkt w czasie (rok, miesiąc, dzień)

**Definicja:**

public class ConcreteDate{

private Long id;

private Date realDate;

private ConcreteDateTemplate concreteDateTemplate;

}

**id:** unikalny klucz konkretnej daty

**realDate:** unikalna data konkretnej daty

**concreteDateTemplate:** Szablon do którego konkretna data należy

## Szablon

Obiekt służy do łączenie poszczególnych dni których plan zajęć jest identyczny w szablony

**Definicja:**

public class ConcreteDateTemplate{

private Long id;

private String name;

private Set<ConcreteDate> concreteDates;

private TermsSet termsSet;

}

**id:** unikalny klucz szablonu

**name:** unikalna nazwa szablonu

**concreteDates:** dni należące do szablonu

**termsSet:** semestr do którego szablon jest podpięty

## Konkretna Lekcja

Obiekt ten przedstawia pojedynczą kratkę na kalendarzu opisaną datą i godziną, do której może być dołączony agregat.

**Definicja:**

public class ConcreteLesson {

private Long id;

private Aggregate aggregate;

private Schedule schedule;

private ClassroomHour classroomHour;

private ConcreteDate concreteDate;

}

**id:** unikalny klucz konkretnej lekcji

**aggregate:** agregatprzypisany do konkretnej lekcji

**schedule:** Szablon do którego został przypisana konkretna lekcja

**classromHour:** godzina odbywania się lekcji w danej sali

**concreteDate:** data odbywania się lekcji

## Wydział

Obiekt przechowuje wydział którego dotyczy plan. Nie jest używany w obecnej wersji systemu

**Definicja:**

public class Department{

private Long id;

private String name;

}

**id:** unikalny klucz wydziału

**name:** unikalna nazwa wydziału

## Kierunek

Obiekt przechowuje kierunek którego dotyczy plan. Nie jest używany w obecnej wersji systemu

**Definicja:**

public class Faculty {

private Long id;

private String name;

}

**id:** unikalny klucz kierunku

**name:** nazwa kierunku

## Oznaczenie grupy dziekanatowej

Obiekt przechowuje znak grupy dziekanatowej

**Definicja:**

public class LetterDesignation{

private int id;

private String name;

}

**id:** unikalny klucz znaku grupy dziekanatowej

**name:** znak grupy dziekanatowej

## Jednostka programowa

Obiekt przechowuje informacje na temat przedmiotu lekcyjnego

**Definicja:**

public class ProgrammeUnit{

private Long id;

private String name;

private ProgrammeUnitType type;

private Set<Aggregate> aggregates;

}

**id:** unikalny klucz jednostki programowej

**name:** nazwa jednostki programowej

**type:** typprowadzonych zajęć (wykład/ćwiczenia/projekt)

**aggregates:** agregaty stworzone do danego przedmiotu

## Typ jednostki programowej

Obiekt przechowuje rodzaj zajęć

**Definicja:**

public class ProgrammeUnitType{

private Long id;

private String name;

}

**id:** unikalny klucz typu jednostki programowej

**name:** nazwa typu jednostki programowej

## Szablon planu lekcji

Definiuje główny szablon planu lekcji stworzony przez użytkownika

**Definicja:**

public class Schedule{

private Long id;

private Set<ConcreteLesson> concreteLessons;

private String name;

private Date date;

private User user;

}

**id:** unikalny klucz szablonu lekcji

**concreteLessons:** konkretne lekcje należące do szablonu

**name:** unikalnanazwa szablonu

**date:** data stworzenia

**user:** właściciel szablonu

## Grupa studencka

Obiekt opisuje dane grupy studenckiej

**Definicja:**

public class StudentGroup{

private Long id;

private Date startDate;

private Date endDate;

private Department department;

private LetterDesignation letterDesignation;

private Faculty faculty;

private Set<Aggregate> aggregates;

}

**id:** unikalny klucz grupy studenckiej

**startDate:** data rozpoczęcia studiów

**endDate:** data zakończenia studiów

**department:** wydział na którym grupa studiuje

**letterDesignation:** znak grupy studenckiej

**faculty:** kierunek na którym studiuje grupa studencka

**aggregates:** agregaty podpięte do grupy studenckiej

## Wykładowca

Obiekt opisujący wykładowcę

**Definicja:**

public class Teacher{

private Long id;

private String name;

private String lastName;

private Avatar avatar;

}

**id:** unikalny klucz wykładowcy

**name:** imię wykładowcy

**lastName**: nazwisko wykładowcy

**avatar:** awatar do którego podpięty jest wykładowca

## Semestr

Obiekt opisuje semestr

**Definicja:**

public class TermsSet {

private Long id;

private Set<ConcreteDateTemplate> concreteDateTemplates;

}

**id:** unikalny klucz semestru

**concreteDateTemplates:** szablony podpięte do semestru

## Użytkownik

Obiekt przechowujący dane na temat użytkownika

**Definicja:**

public class User {

private Long id;

private String login;

private String password;

private String passwordMd5;

private Set<Authorization> authorizations;

private boolean isLoggedIn;

}

**id:** unikalny klucz użytkownika

**login:** login użytkownika

**password:** hasło użytkownika

**passwordMd5:** zaszyfrowane hasło użytkownika

**authorizations:** autoryzacje przypisane do użytkownika

**isLoggedIn:** flaga zalogowania

# Architektura

Rozdział ten opisuje architekturę systemu, poczynając od ogólnego opisu, kończąc na szczegółowym opisie poszczególnych modułów

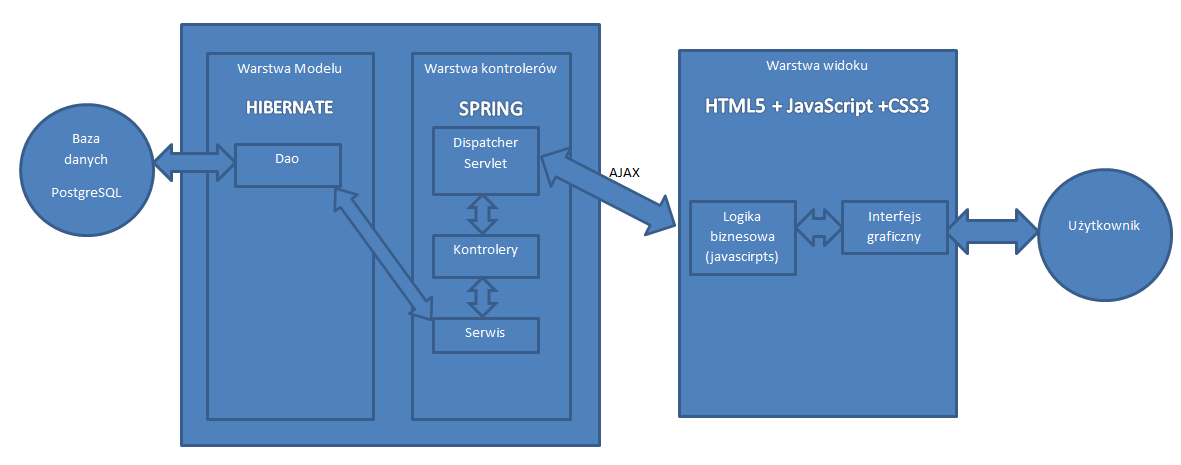
## Wprowadzenie

System jest przeznaczony do działania na serwerze w sieci lokalnej lub globalnej gdzie użytkownicy mają do niego dostęp. System w całości działa na pojedynczym serwerze i nie potrzebuje połączenia z żadnymi innym serwerami, dlatego może być stosowany w zamkniętych sieciach.

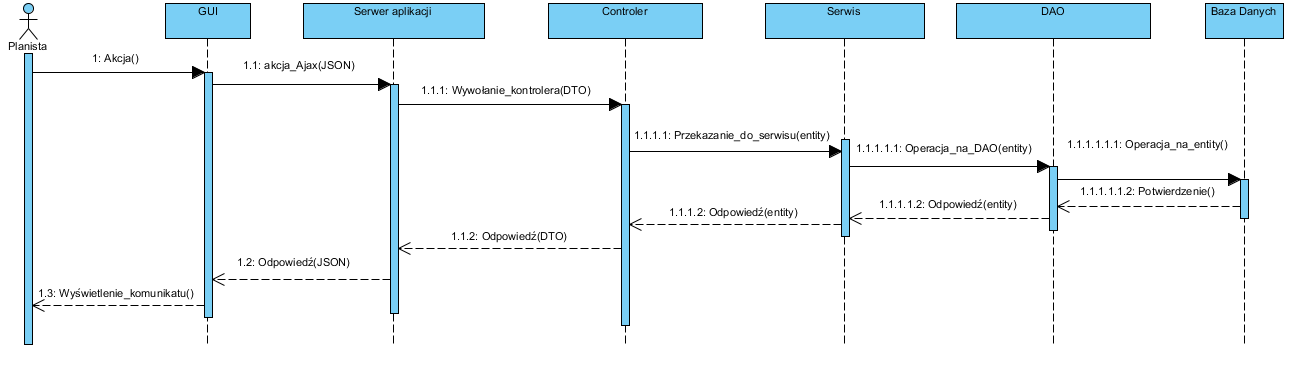
## Komunikacja wysokopoziomowa

Komunikacja pomiędzy modułami frontendu i backendu opiera się na przesyłaniu komunikatów AJAX. W zależności od treści zapytania w procedurze obsługi występują nieznaczne różnice. Ogólny szablon drogi komunikatu przedstawiony jest na rysunku 1, podczas gdy diagram sekwencyjny pokazany jest na rysunku 2. Całą procedurę można przedstawić za pomocą listy podmodułów do których trafia po kolei komunikat:

1. *Moduł http – javascript* – jeden z skryptów tworzy zapytanie AJAX i przesyła dane w formacie JSON (nie w każdym zapytaniu są przesyłane JSON).
2. *Moduł spring* – *serwer aplikacji* – zapytanie jest odbierane i mapowane na obiekt Java (DTO), który jest następnie kierowany przez *dispatcher servlet* do odpowiedniego kontrolera
3. *Moduł spring* – *kontroler* – kontroler odbiera obiekt DTO i mapuje go na obiekt *entity* i kieruje do odpowiedniego serwisu
4. *Moduł spring* – *serwis* – serwis kieruje *entity* do warstwa DAO
5. *Moduł hibernate* - *DAO* – warstwa DAO operuje na obiekcie *entity*, czyli w efekcie na bazie danych
6. Wynik zapytanie jest przekazywany z powrotem, mapowany na obiekt DTO przez kontroler, następnie na JSON przez serwer aplikacji i zwracany do skryptu który stworzył zapytanie.



Rysunek : Ścieżka zapytania po modułach przedstawiona za pomocą strzałek



Rysunek : Diagram sekwencyjny przepływu wiadomości

## HTML

Struktura:

* AghPlanner/src/main/resources/pages/ – główny folder
  + index.html – strona logowania
  + main.html – główna strona
  + css/ - folder z plikami css
  + js/ - folder z skryptami
    - model/ - folder z prototypami
  + libs/ - folder z bibliotekami

Jedyny widoczny dla użytkownika moduł. Jego głównym zadaniem jest budowa graficznego interfejsu z którym użytkownik może wchodzić w interakcję, obsługa logowania, oraz wysyłanie i odbieranie zapytań typu AJAX od modułu Spring.

Obsługa logowania rozwiązana jest przez technologie Spring Security. Jest ona zaimplementowana w pliku index.html, który jest pierwszą wyświetlaną stroną i w razie powodzenia w logowaniu, użytkownik kierowany jest na stronę main.html.

Strona main.html jest główną stroną projektu na której znajduje się kalendarz i użytkownik działa tylko w ramach tej strony lub jej kopii tworzonych w ramach widoków.

Folder js/ zawiera skrypty obsługujące działanie strony oraz folder model/ w którym znajdują się prototypy obiektów tworzonych przez JavaScript takich, jak awatary czy zasobniki.

Komunikacja oparta jest na technologii AJAX, która dzięki temu że jest komunikacją asynchroniczną pozwala na przeładowywanie tylko części dokumentu, zamiast przesyłania całej strony Html.

Przykładowe zapytanie dla agregatów:

$.ajax({

url: "/aggregates",

type: 'GET',

dataType: 'json',

contentType: 'application/json',

mimeType: 'application/json',

success: function(data) {

data.forEach(function(entry) {

aggregateContainer.addAggregate(

new Aggregate(entry));

});

}

});

## Spring

Struktura modułu Spring:

* AghPlanner/src/main/
  + java/pl/edu/agh/planner/
    - config/
      * Config.java – plik konfiguracyjny
    - dto/ - folder z obiektami DTO
    - controllers/ – folder z kontrolerami
    - service/ - folder z serwisami
    - Application - plik rozruchowy
  + resources/
    - application.properties -plik konfiguracyjny

Moduł Spring zbudowany jest za pomocą frameworku Spring i jego głównym zadaniem jest zapewnienie komunikacji pomiędzy Modułem Hibernate oraz Html, poprzez mapowanie informacji pomiędzy nimi przechodzącymi oraz zapewnienie szablonu projektu.

Moduł Spring został stworzony przy użyciu pluginu spring boot. Z tego powodu dispatcher servlet jest generowany automatycznie, dlatego jego obsługa nie została pokryta w tej dokumentacji.

Przychodzące zapytania AJAX przetworzone przez serwer aplikacji są obsługiwane przez kontrolery znajdujące się w folderze:

* AghPlanner/src/main/java/pl/edu/agh/planner/controllers/

Kontrolery są oznaczone adnotacją @RestController co pozwala im używać adnotacji @RequestMapping() w celu zapewnienia mapowania elementów JSON na obiekty Java.

W celu mapowania DTO na entity oraz w drugą stronę entity na DTO wywoływana jest klasa DtoEntityMapper znajdująca się w folderze:

* AghPlanner/src/main/java/pl/edu/agh/planner/utils/

Do operowania na obiektach DAO służą serwisy zlokalizowane w folderze

* AghPlanner/src/main/java/pl/edu/agh/planner/service/

Serwisy oznaczone są adnotacją @Service i używają interfejsu *ServiceInterface*

Do uruchamiania aplikacji służy plik Application zawierający klasę main oraz Adnotacje @SpringBootApplication w celu uruchomienia auto-konfiguracji

Plik konfiguracyjny *application.properties* zawiera konfiguracje dla serwera aplikacji

## Moduł Hibernate

Struktura modułu Hibernate:

* AghPlanner/src/main/
  + java/pl.edu.agh.planner/
    - dao/ - folder z obiektami DAO
    - domain - folder z obiektami entity
    - utils/
      * GenericQuery.java - plik z transakcjami
      * HibernateUtil.java - plik z fabryką sesji
  + resources/
    - Hibernate.cfg.xml - plik konfiguracyjny

Moduł Hibernate jest odpowiedzialny za mapowanie relacyjnej bazy danych na obiekty oraz realizacje wszystkich zleconych operacji na bazie danych.

Obiekty DAO zapewniają jednolity interfejs do operacji na bazie danych i wykonują operacje na obiektach *entity*.

Obiekty *entity* są zmapowanymi rekordami z bazy danych. Plik konfiguracyjny Hibernate.cfg.xml pozwala na zmianę bazy danych i dostępowych danych logowania do niej, zapisane są też w nim wszystkie obiekty mapowania entity.

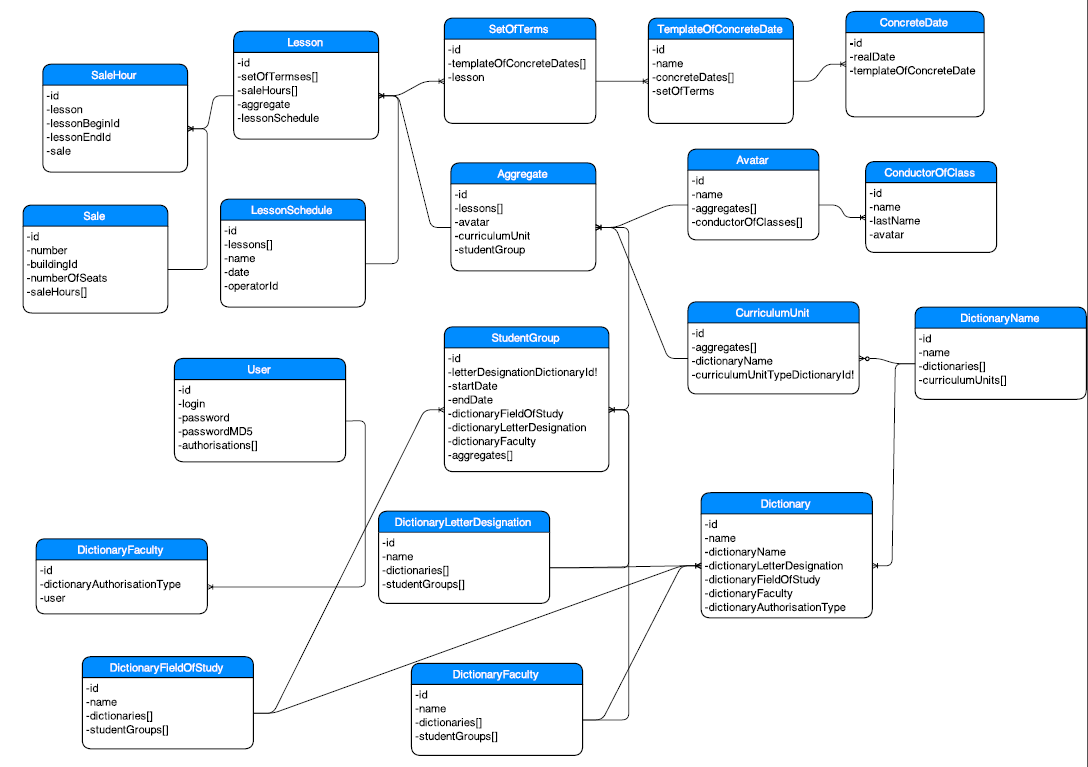
## Baza Danych

Baza danych została stworzona przy użyciu systemu bazodanowego PostgreSQL.

Tabele umieszczone w bazie danych i są odzwierciedleniem struktur opisanych w rozdziale obiekty. Wyjątek stanowią tabele połączeniowe zastosowane w połączeniu wiele do wielu.

Baza danych została rozbudowana o dodatkowe tabele nie używane w obecnej wersji, tabele są przygotowane pod dalszą rozbudowę programu.

Diagram bazy danych jest ukazany na rysunku 3.

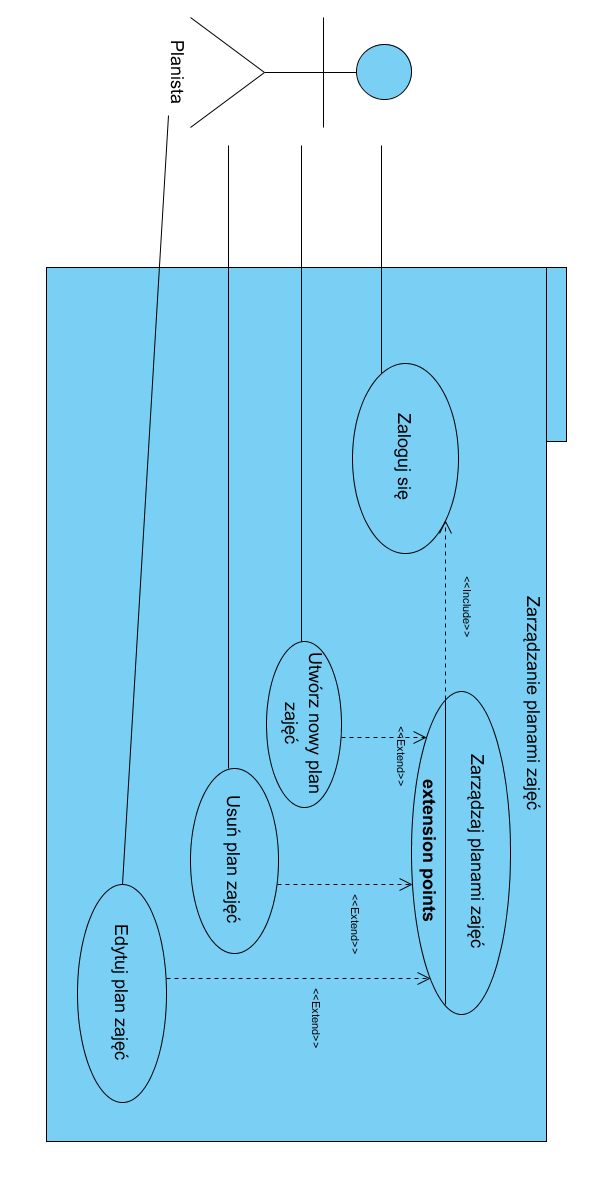


Rysunek : Diagram relacji bazy danych

Baza danych odzwierciedla obiekty które zostały opisane w rozdziale Obiekty.

# 4. Diagramy przypadków użycia

## 4.1. Zarządzanie planami zajęć



Rysunek : Diagram przypadku użycia zarządzania planami zajęć

LOGOWANIE:

Aktorzy: Planista

Wyzwalacz: Planista naciska na przycisk „Zaloguj się”

Warunek początkowy: Planista nie jest zalogowany

Przepływ podstawowy:

1. Wprowadzanie adresu e-mail i hasła

2. Komunikat o udanym logowaniu

3. Udostępnienie interfejsu użytkownika

Przepływ alternatywny:

1.1. Błędne dane logowania

1.2. Komunikat o błędzie

1.2.1.1. Ponowne logowanie

TWORZENIE NOWEGO PLANU ZAJĘĆ:

Aktorzy: Planista

Wyzwalacz: Planista naciska na przycisk „Stwórz nowy plan zajęć”

Warunek początkowy: Planista jest zalogowany

Przepływ podstawowy

1. Wybranie nazwy oraz okresu obowiązywania planu.

2. Planista przypisuję odpowiednią grupę studencką.

3. Planista klika „Zapisz” w celu zapisania planu do którego potem będzie mógł przypisywać

terminy z zajęciami.

Przepływ alternatywny:

1.1 Błędna nazwa pliku

1.2 Błędne dane- komunikat o błędzie

EDYCJA PLANU ZAJĘĆ:

Aktorzy: Planista

Wyzwalacz: Planista naciska na przycisk „Edytuj plan zajęć”

Warunek początkowy: Planista jest zalogowany

Przepływ podstawowy

1. Wybranie planu do edycji.

2. Zatwierdzenie zmian.

Przepływ alternatywny:

1.1 Wybranie planu którego nie można edytować

1.2 Wprowadzenie błędnych danych- komunikat o błędzie

USUWANIE PLANU ZAJĘĆ:

Aktorzy: Planista

Wyzwalacz: Planista naciska na przycisk „Usuń plan zajęć”

Warunek początkowy: Planista jest zalogowany

Przepływ podstawowy:

1. Wybranie planu do usunięcia

2. Zatwierdzenie usunięcia

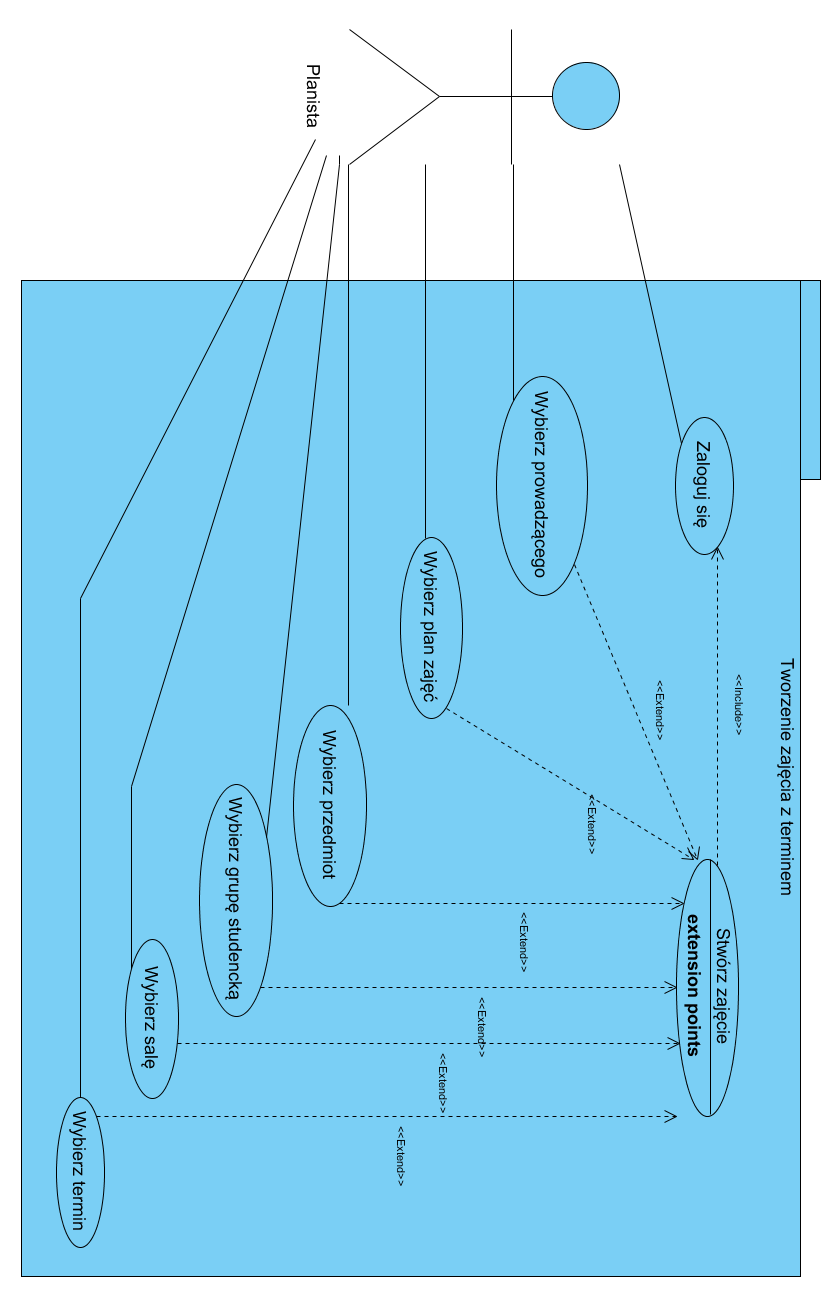
Przepływ alternatywny:

1.1 Wybranie planu który już nie istnieje(jest usunięty a strona jest nie odświeżona).

1.2 Komunikat o błędzie.

1.2 Zarządzanie przedmiotami

## 4.4 Tworzenie zajęcia wraz z terminem



Rysunek : Diagram przypadku użycia zarządzania zajęciami z terminami

TWORZENIE ZAJĘCIA:

Aktorzy: Planista

Wyzwalacz: Planista naciska na przycisk „Agregaty”

Warunek początkowy: Planista jest zalogowany oraz ma otwarte wszystkie zasobniki

Przepływ podstawowy:

1. Planista przeciąga awatara z zasobnika awatarów na puste miejsce w zasobniku agregatów

2. Planista przeciąga jednostkę programu studiów z zasobnika jednostek programu studiów na agregat w zasobniku agregatów

3. Planista przeciąga grupę studencką z zasobnika grup studenckich na agregat w zasobniku agregatów

4. Planista umieszcza agregat na kalendarzu

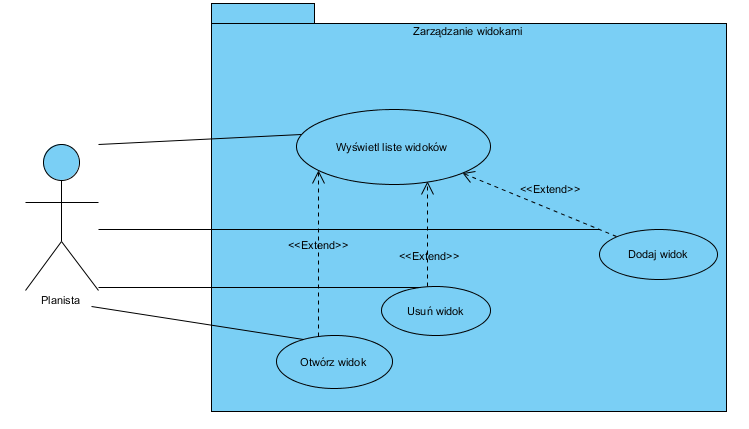
5. Planista klika na umieszczonym agregacie

6. Planista wybiera sale do której przypisany ma być agregat

Przepływ alternatywny:

1.3 Komunikaty o ewentualnych błędach.

## 4.6 Zarządzanie widokami



Rysunek : Diagram przypadków zarządzania widokami

Otworzenie widoku:

Aktorzy: Planista

Wyzwalacz: Planista naciska na przycisk „Widok”

Warunek początkowy: Planista jest w głównym ekranie

Przepływ podstawowy:

1. Planista wybiera jeden z zapisanych widoków w celu otworzenia go w aktualnym oknie

Przepływ alternatywny

1.1 Nie istnieje żaden zapisany widok

Zapisanie widoku:

Aktorzy: Planista

Wyzwalacz: Planista naciska na przycisk „Widok”

Warunek początkowy: Planista jest w głównym ekranie

Przepływ podstawowy:

1. Planista wypiera opcję „Zapisz nowy widok”
2. Planista wpisuje nazwę widoku
3. Planista naciska przycisk „Zapisz”

Przepływ alternatywny

4.1 Widok o takiej już istnieje. Widok nie zostaje zapisany

Usuwanie widoku:

Aktorzy: Planista

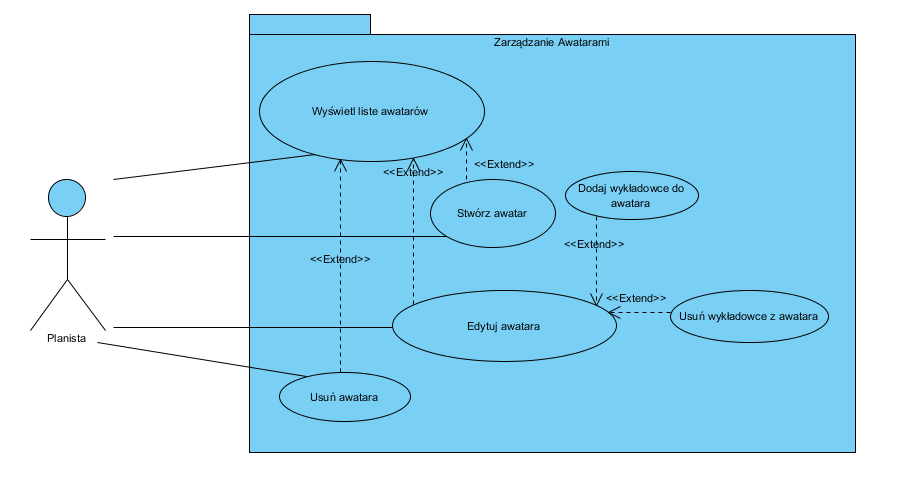
Wyzwalacz: Planista naciska na przycisk „Widok”

Warunek początkowy: Planista jest w głównym ekranie

Przepływ podstawowy:

1. Planista wypiera widok do usunięcia
2. Planista naciska przycisk usuń

## 4.8 Zarządzanie awatarami



Rysunek : Diagram przypadku użycia zarządzania awatarami

Wyświetlenie listy awatarów:

Aktorzy: Planista

Wyzwalacz: Planista naciska na przycisk „Awatarzy”

Warunek początkowy: Planista jest w oknie głównym oraz zalogowany

Przepływ podstawowy:

1. Planista naciska przycisk awatary

Stworzenie awatara:

Aktorzy: Planista

Wyzwalacz: Planista naciska na przycisk „Awatary”

Warunek początkowy: Planista jest zalogowany oraz zasobnik awatarów oraz zasobnik wykładowców jest włączony

Przepływ podstawowy:

1. Planista zaczyna przeciągnąć wykładowcę
2. Planista upuszcza wykładowcę na puste miejsce w zasobniku awatarów

Usuwanie awatara:

Aktorzy: Planista

Wyzwalacz: Planista naciska na przycisk „Awatary”

Warunek początkowy: Planista jest zalogowany oraz zasobnik awatarów jest włączony

Przepływ podstawowy:

1. Planista zaczyna przeciągać wykładowcę przypisanego do danego awatara
2. Planista upuszcza wykładowcę poza zasobnikiem
3. Planista kontynuuje przeciąganie i upuszczanie wykładowców aż awatar pozostanie pusty

Edytowanie awatara:

Aktorzy: Planista

Wyzwalacz: Planista naciska na przycisk „Awatary”

Warunek początkowy: Planista jest zalogowany oraz zasobnik awatarów oraz zasobnik wykładowców jest włączony

Przepływ podstawowy:

1. Planista zaczyna przeciągnąć wykładowcę
2. Planista upuszcza wykładowcę na awatara w zasobniku awatarów
3. Wykładowca zostaje dodany do awatara

Przepływ alternatywny:

1. Planista zaczyna przeciągać wykładowcę przypisanego do danego awatara
2. Planista upuszcza wykładowcę poza zasobnikiem
3. Wykładowca zostaje usunięty z awatara

# Instalacja

Instalacja systemu jest procesem prostym z uwagi na sposób dostarczenia projektu. System może być zainstalowany na dowolnym komputerze, jednak z powodu funkcjonalnego najlepiej żeby był zainstalowany na serwerze do którego występuje dostęp z sieci.

Istnieją dwa sposoby instalacji systemu:

## Sposób I

* **Instalacja bazy PostreSQL ver. 8.3.7**

System bazodanowy można ściągnąć ze strony <http://www.postgresql.org.pl/>.

Wykonujemy następujące komendy aby zainstalować bazę:

**tar xvfz postgresql-8.3.7.tar.gz**

**cd postgresql-8.3.7**

**./configure**

**Make**

**Make install**

* **Stworzenie bazy PostreSQL**

System wymaga stworzenia bazy danych aby poprawnie działać.

Wykonujemy następujące komendy:

Tworzymy użytkownika:

**adduser postgres**

**passwd password**

**Tworzymy bazę danych:**

**createdb postgres**

* **Instalacja Maven 3**

Program maven może być pobrany z adresu: <https://maven.apache.org/>. Jego instalacja jest pokryta w poradniku dostępnym pod tym samym adresem dlatego nie zostaje pokryta w tym dokumencie

* **Ściągnięcie systemu z repozytorium git**

Należy ściągnąć system z repozytorium git dostępnym pod adresem <https://github.com/agh-ki-io/planner.git> co można osiągnąć instalując program do repozytoriów git lub ściągając bezpośrednio ze strony archiwum zip

* **Skompilowanie repozytorium**

W celu skompilowania repozytorium należy włączyć program do obsługi poleceń

W systemie Windows: Uruchom>cmd

W systemie Linux: Terminal

Przechodzimy do katalogu AghPlanner i wykonujemy polecenie

**mvn clean install**

* **Uruchomienie systemu**

Przechodzimy do katalogu target utworzonego przez poprzednią komendę i uruchomimy program instrukcją:

**java -jar AghPlanner.jar**

## Sposób II

* **Instalacja serwera tomcat**

Do działania systemu potrzeby jest zainstalowanie serweru tomcat. Aplikacja jest dostępna za darmo pod adresem <http://tomcat.apache.org/>.

* **Instalacja bazy PostreSQL ver. 8.3.7**

System bazodanowy można ściągnąć ze strony <http://www.postgresql.org.pl/>.

Wykonujemy następujące komendy aby zainstalować bazę:

**tar xvfz postgresql-8.3.7.tar.gz**

**cd postgresql-8.3.7**

**./configure**

**Make**

**Make install**

* **Stworzenie bazy PostreSQL**

System potrzebuje stworzenia bazy danych aby poprawnie działać.

Wykonujemy następujące komendy:

Tworzymy użytkownika:

**adduser postgres**

**passwd postgres**

**Tworzymy bazę danych:**

**createdb test**

* **Ściągnięcie systemu z repozytorium git**

Należy ściągnąć system z repozytorium git dostępnym pod adresem <https://github.com/agh-ki-io/planner.git> co można osiągnąć instalując program do repozytoriów git lub ściągając bezpośrednio ze strony archiwum zip

* **Skompilowanie repozytorium**

**System należy skompilować w IDE Intellij dostępnym pod adresem** <https://www.jetbrains.com/idea/> do którego należy zaimportować repozytorium i uruchomić. Poradniki jak obsługiwać Intellij dostępne są pod powyższą stroną inie sa pokryte w tej dokumentacji.

* **Umieszczamy system na serwerze**

Aby umieścić system na serwerze należy przekopiować pliki systemu do katalogu /**war**

# Materiały źródłowe

<http://dev.w3.org/html5/html-author/> - Dokumentacja standardu html5

<http://www.w3.org/TR/#tr_CSS> – Dokumentacja standardu CSS3

<http://hibernate.org/> - Główna strona frameworku Hibernate

<https://spring.io/> - Główna strona frameworku Spring

<https://www.jetbrains.com/idea/> - Głowna strona IDE Intellij

<https://github.com/> - Główna strona repozytoriów git

<http://www.postgresql.org.pl/> - Główna strona systemu bazodanowego PostreSQL

<http://tomcat.apache.org/> - Główna strona serwera apache tomcat

<https://maven.apache.org/> - Głowna strona programu Maven

# Spis ilustracji

[Rysunek 1: Ścieżka zapytania po modułach przedstawiona za pomocą strzałek 14](#_Toc423272395)

[Rysunek 2: Diagram sekwencyjny przepływu wiadomości 15](#_Toc423272396)

[Rysunek 3: Diagram realcji bazy danych 18](#_Toc423272397)

[Rysunek 4: Diagram przypadku użycia zarządzania planami zajęć 19](#_Toc423272398)

[Rysunek 5: Diagram przypadku użycia zarządzania zajęciami z terminami 21](#_Toc423272399)

[Rysunek 6: Diagram przypadków zarządzania widokami 22](#_Toc423272400)

[Rysunek 7: Diagram przypadku użycia zarządzania awatarami 24](#_Toc423272401)