



# Dokumentacja TO

Łukasz Pitrus, Grzegorz Gackowski

Dawid Białka, Krzysztof Retkiewicz

Temat projektu: Klub fitness

Grupa czwartek 14:40

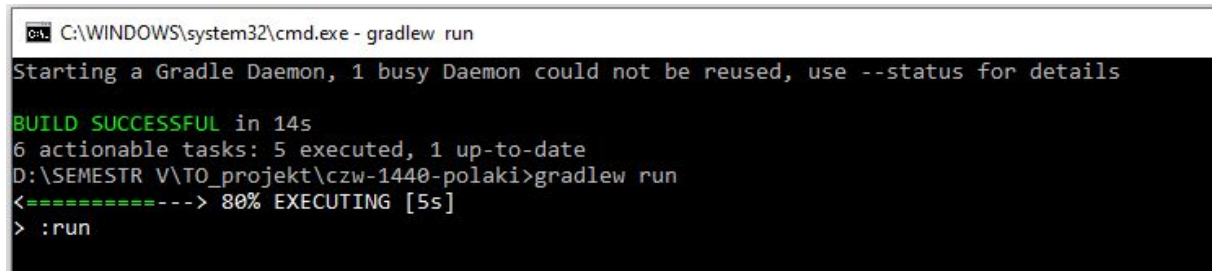
## Uruchomienie projektu.

### 1. Uruchomienie projektu na Windowsie.

Otwieramy wiersz polecenia, wchodzimy do folderu z projektem i wpisujemy: `start gradlew.bat build`.

```
D:\SEMESTR V\TO_projekt\czw-1440-polaki>start gradlew.bat build
```

Otworzy się nam drugi wiersz polecenia w którym wpisujemy `gradlew run`.



```
C:\WINDOWS\system32\cmd.exe - gradlew run
Starting a Gradle Daemon, 1 busy Daemon could not be reused, use --status for details

BUILD SUCCESSFUL in 14s
6 actionable tasks: 5 executed, 1 up-to-date
D:\SEMESTR V\TO_projekt\czw-1440-polaki>gradlew run
<=====--> 80% EXECUTING [5s]
> :run
```

### 2. Uruchomienie projektu na Linuxie.

Aby uruchomić projekt na komputerze z systemem Linux należy otworzyć terminal i wejść do folderu z projektem.

Może być wymagane nadanie odpowiedniemu plikowi praw do wykonywania.

```
→ czw-1440-polaki git:(m1) sudo chmod +x gradlew
```

Następnie wykonać polecenie:

```
→ czw-1440-polaki git:(m1) X ./gradlew run
```

Program powinien uruchomić się w nowym oknie.

## Odpowiedzialności.

Dawid Białka, Łukasz Pitrus.

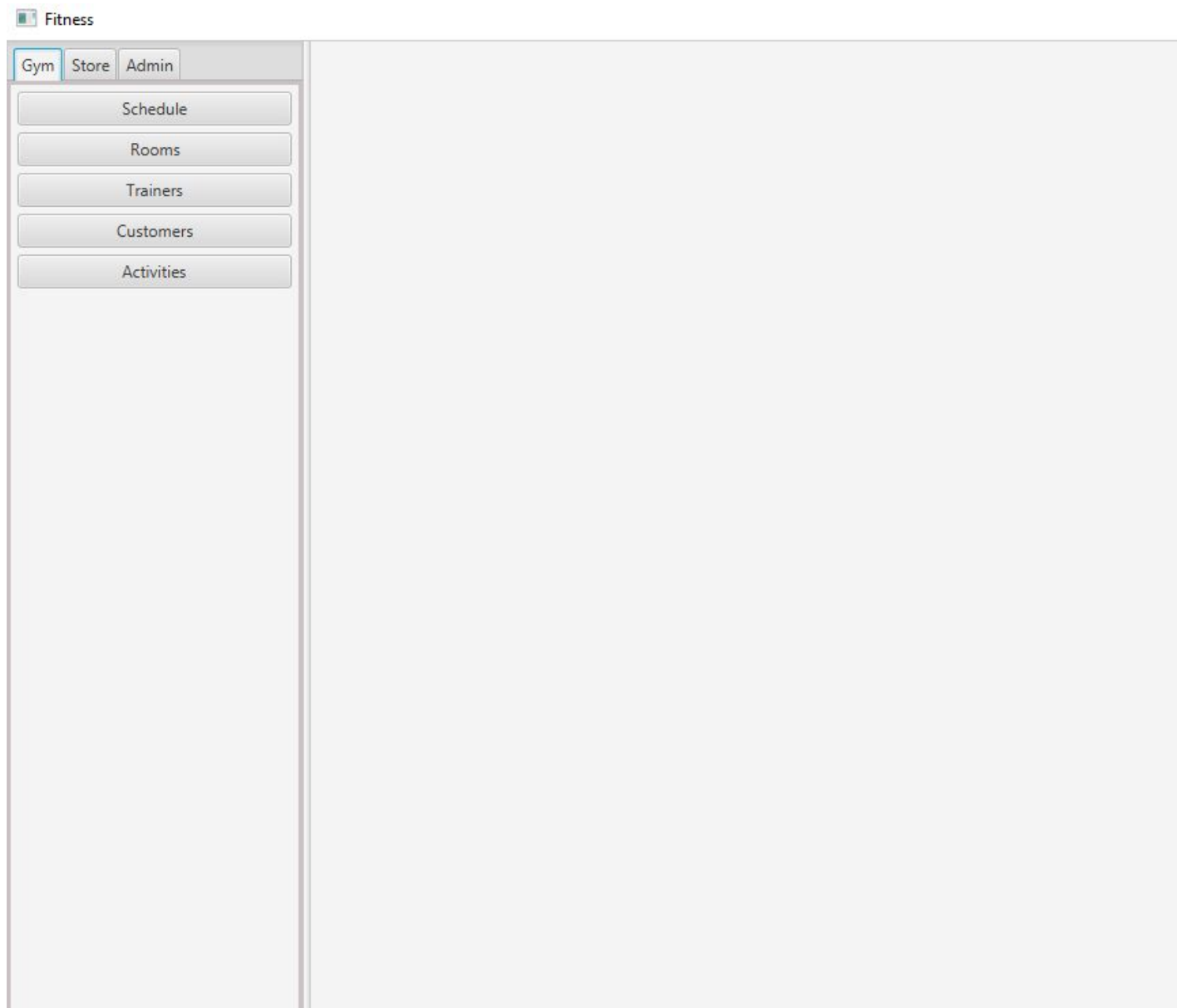
Backend. Wszystkie klasy związane z modelem, zapisywaniem, odczytywaniem, walidacją danych.

Grzegorz Gackowski, Krzysztof Retkiewicz.

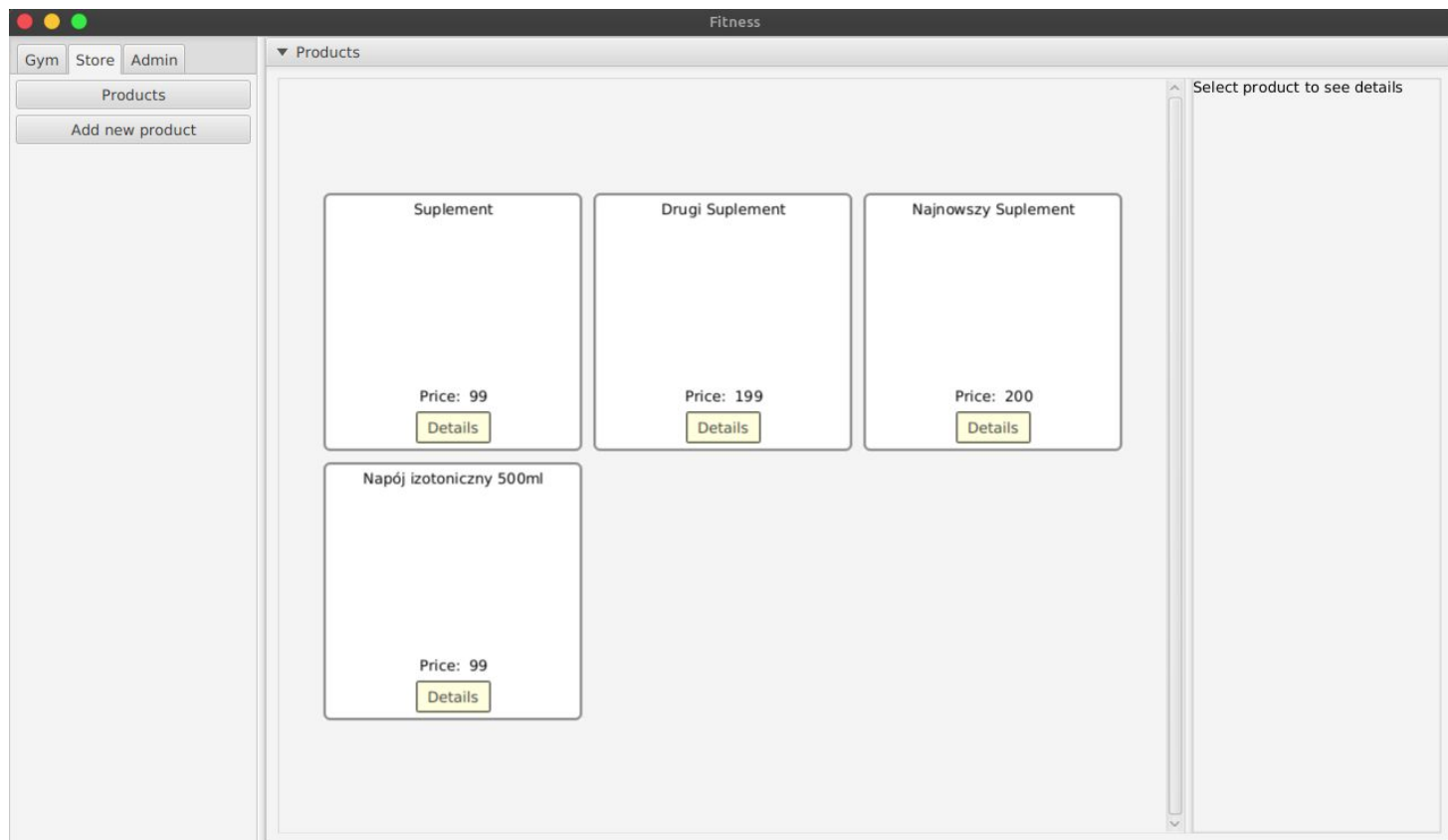
Frontend. Wszystkie klasy związane z widokiem, kontrolerami, aktualizowaniem widoku i modelu na podstawie akcji użytkownika.

## Działanie programu.

Okienko główne programu.



Podgląd produktów



Dodawanie nowego produktu wraz z walidacją poprawności wprowadzonych danych i wyświetleniu komunikatu o błędnie wprowadzonych danych.

**Fitness**

Gym Store Admin

Products

Add new product

▼ Add new product to the store

Name

Price

Quantity

Img

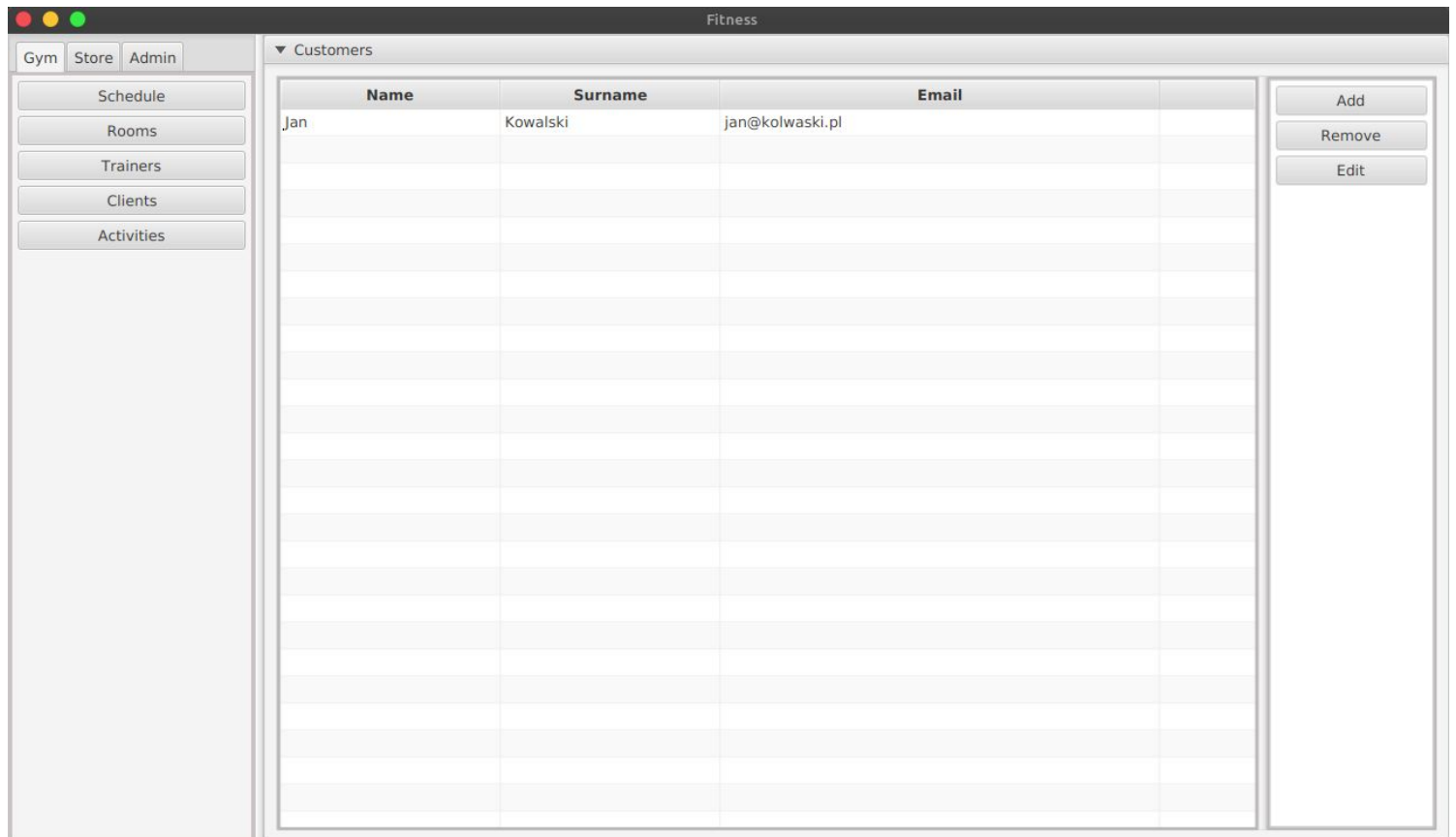
Description

Name cannot be empty

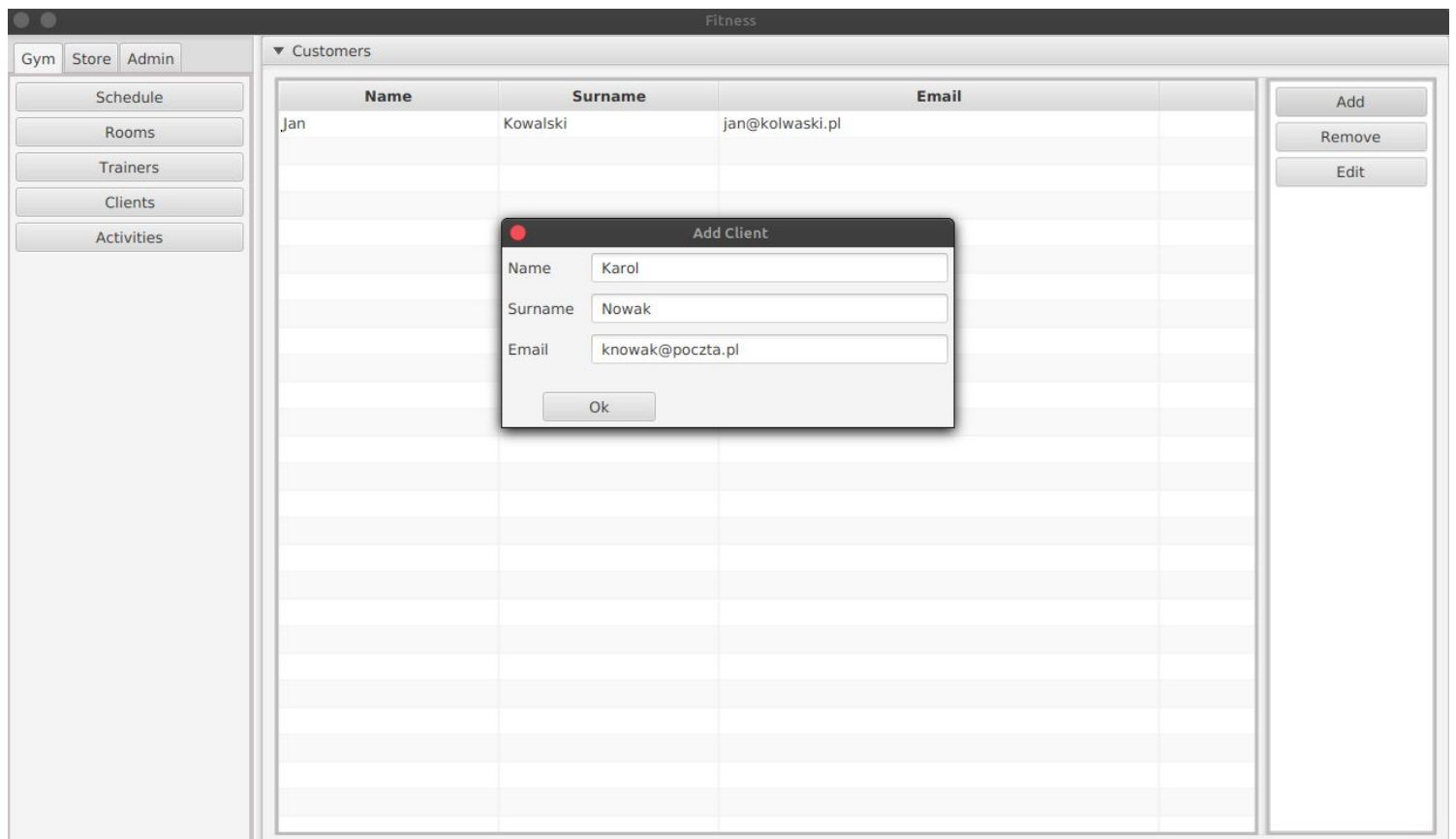
Description cannot be empty

Quantity must be a number

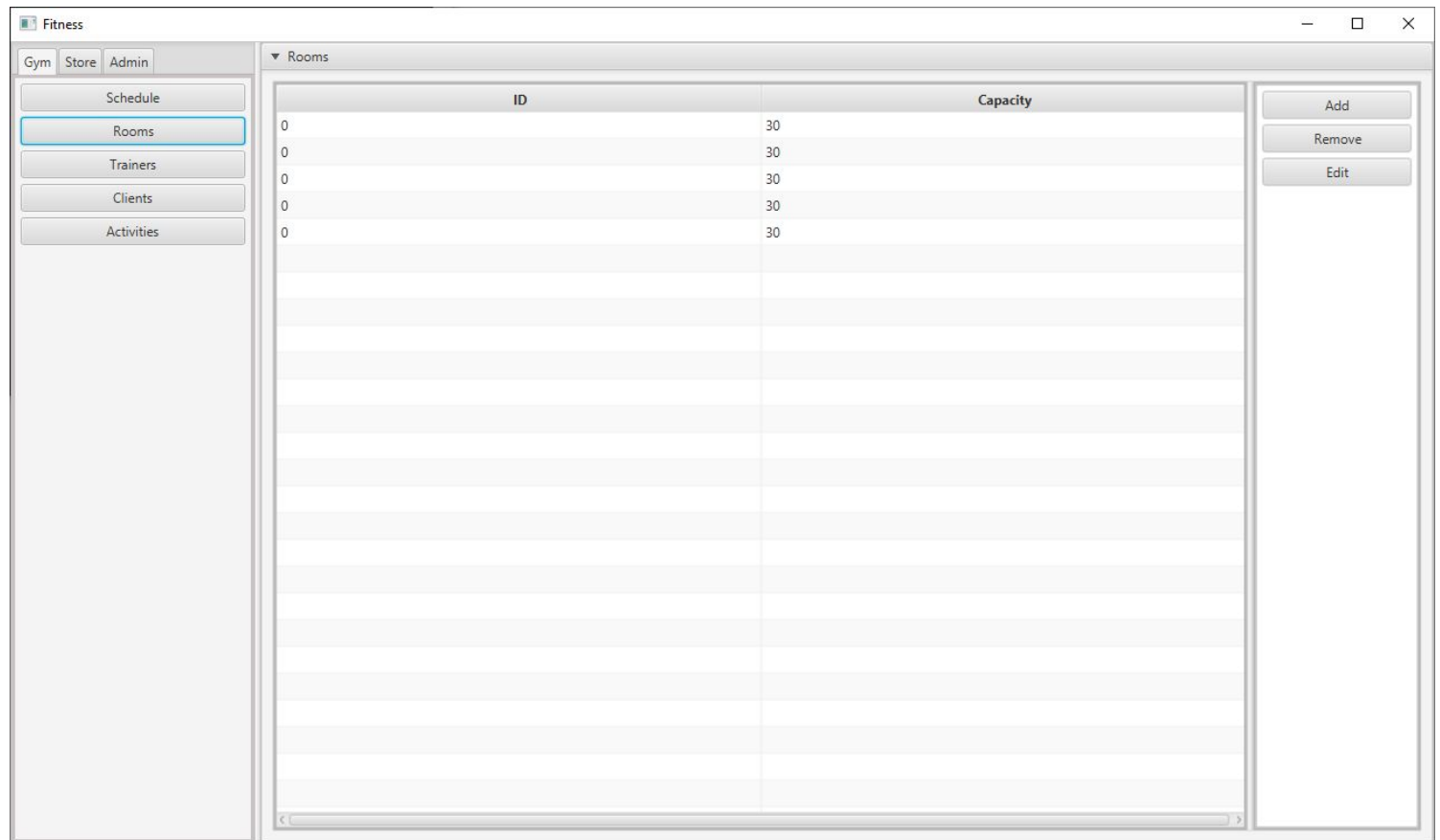
## Podgląd klientów



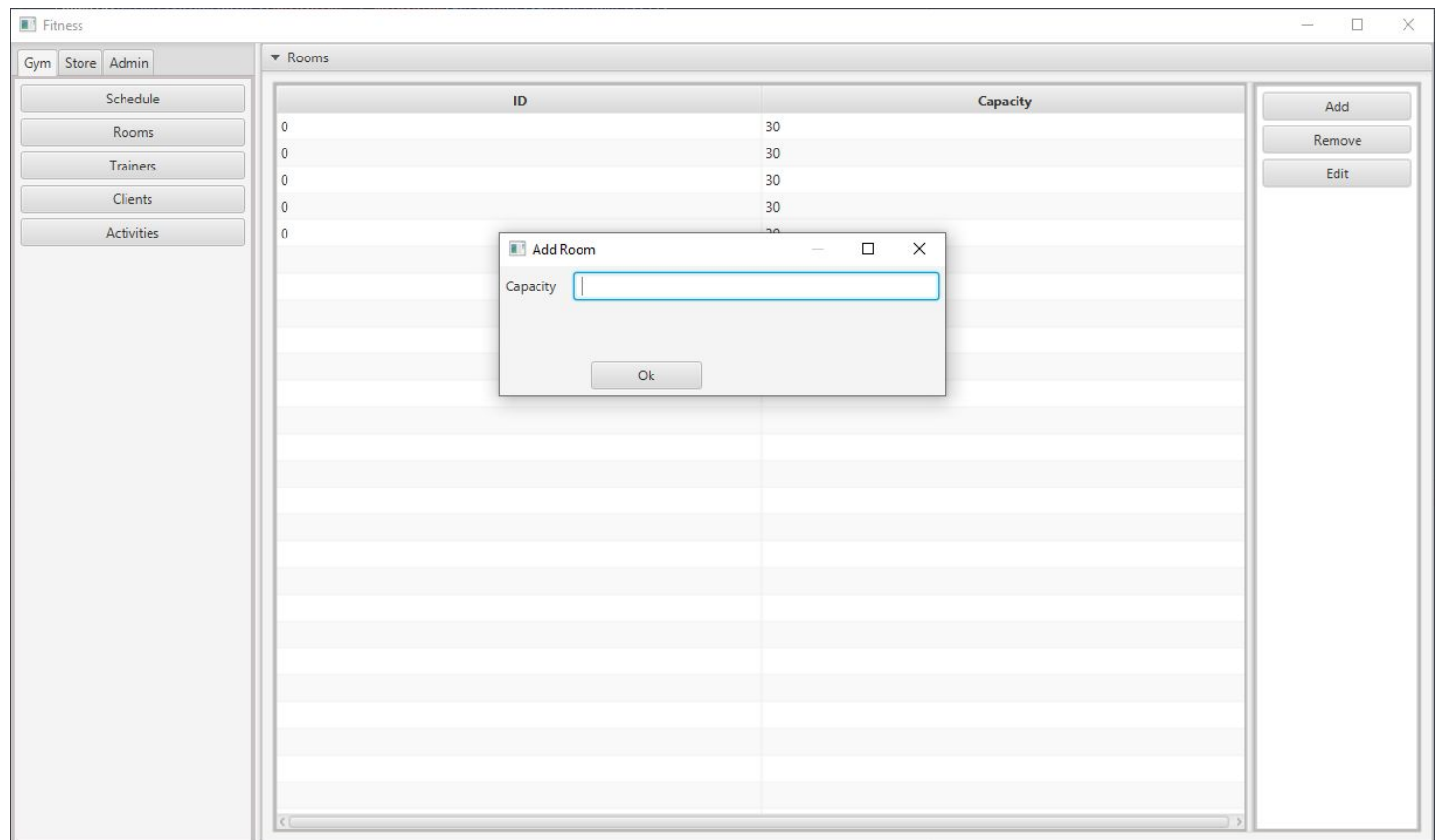
## Dodawanie klienta



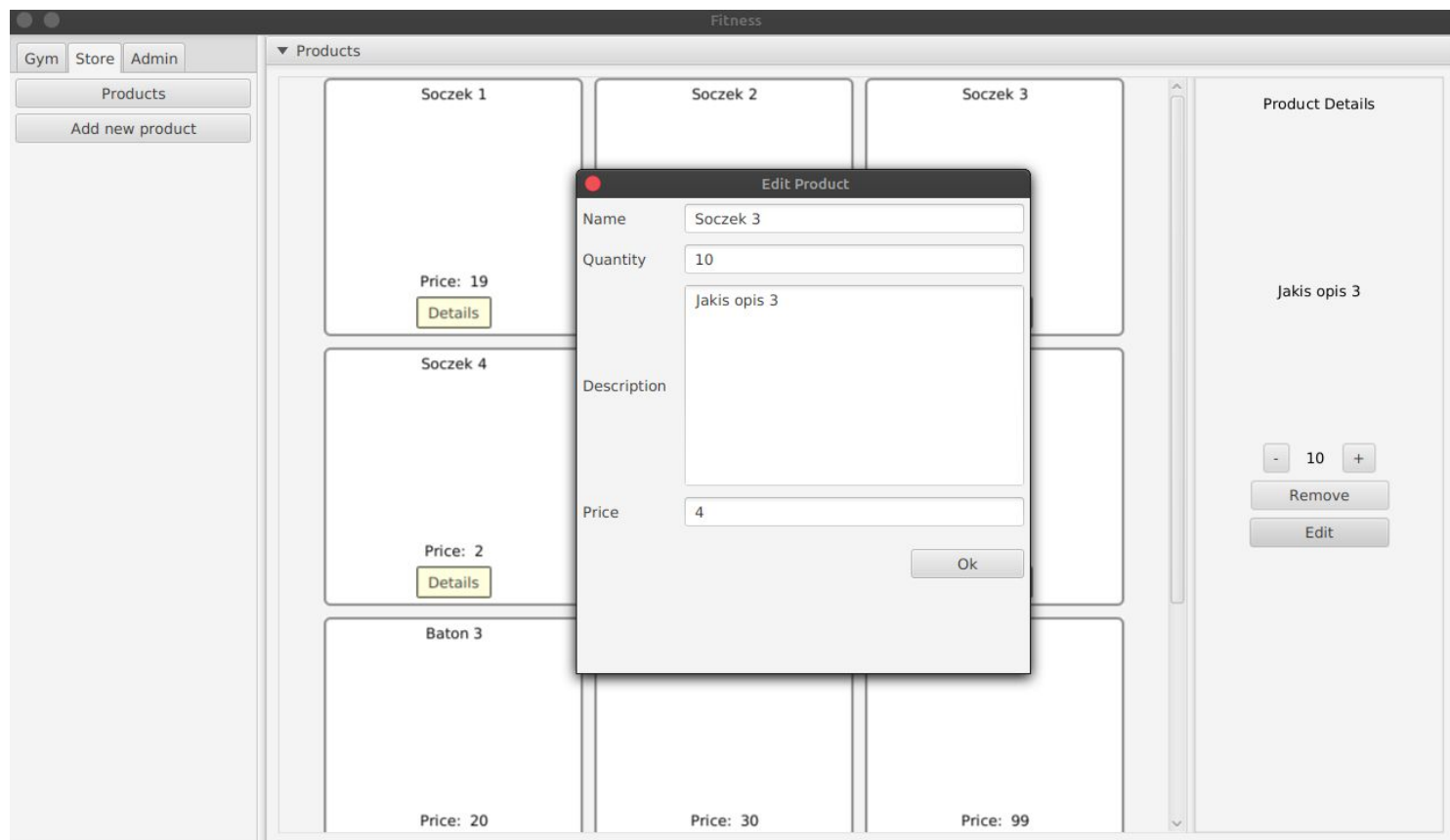
Podgląd sal:



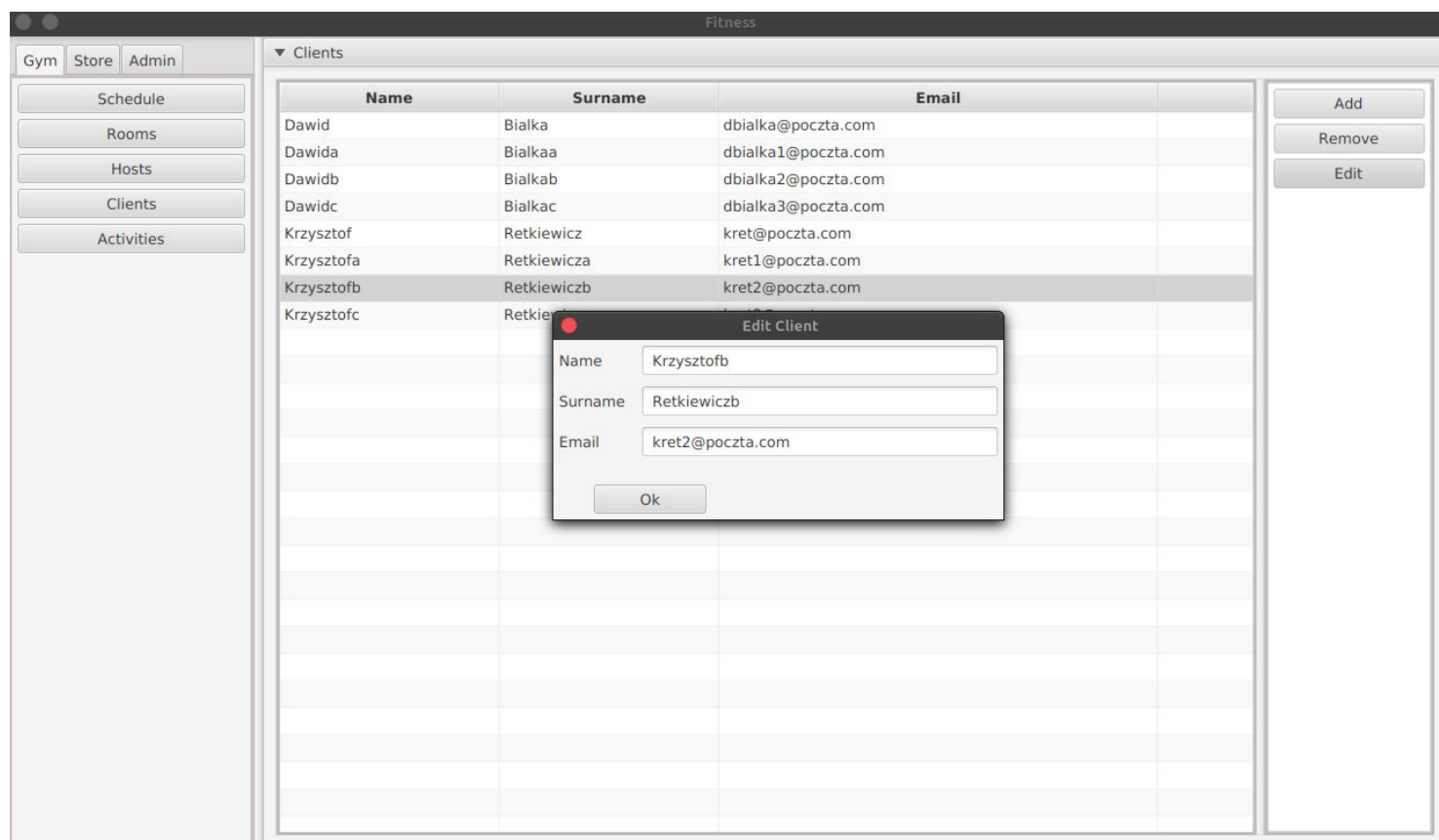
Dodawanie sali:



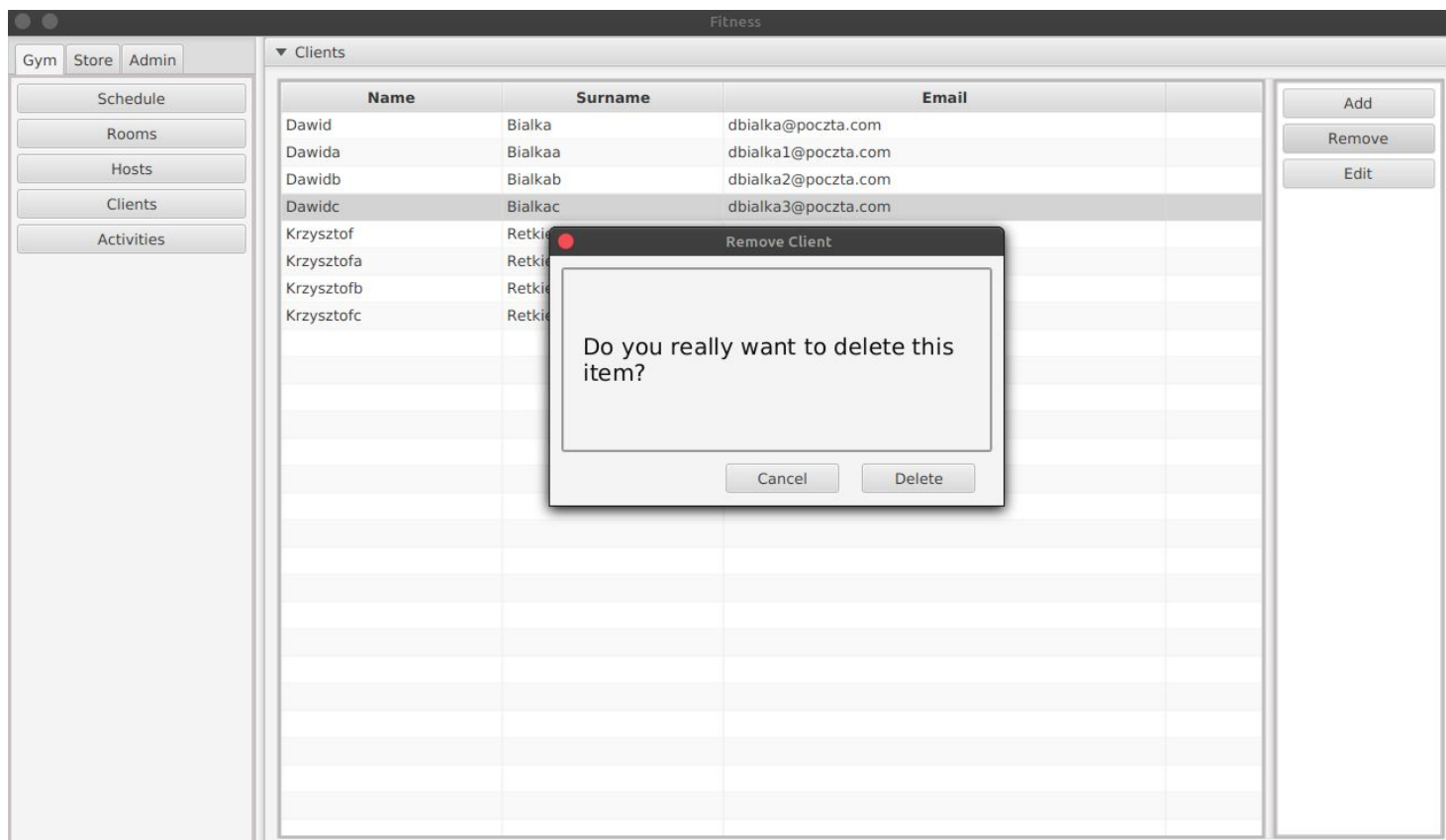
## Edycja produktu:



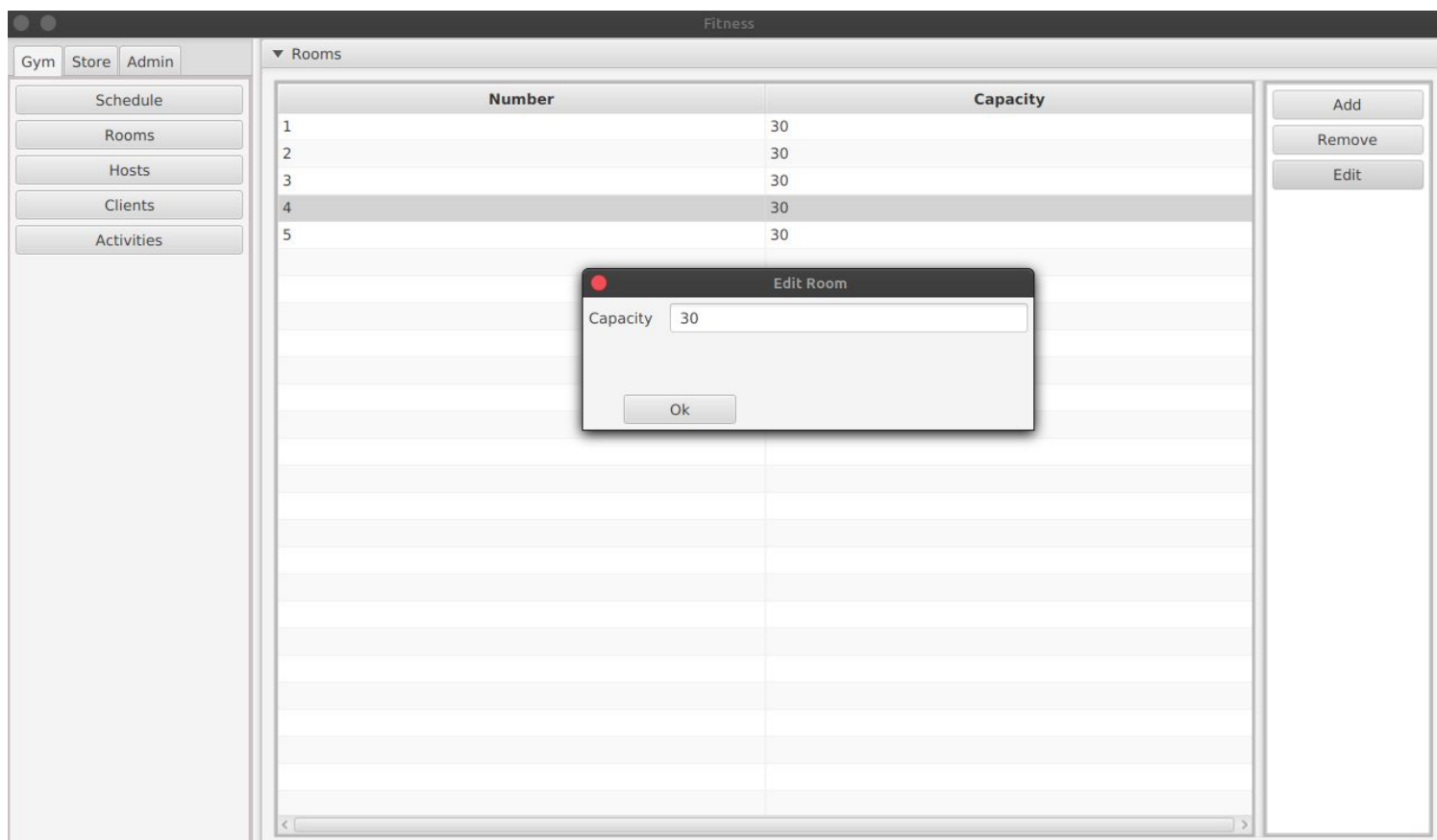
## Edycja klienta:



## Usuwanie klienta:

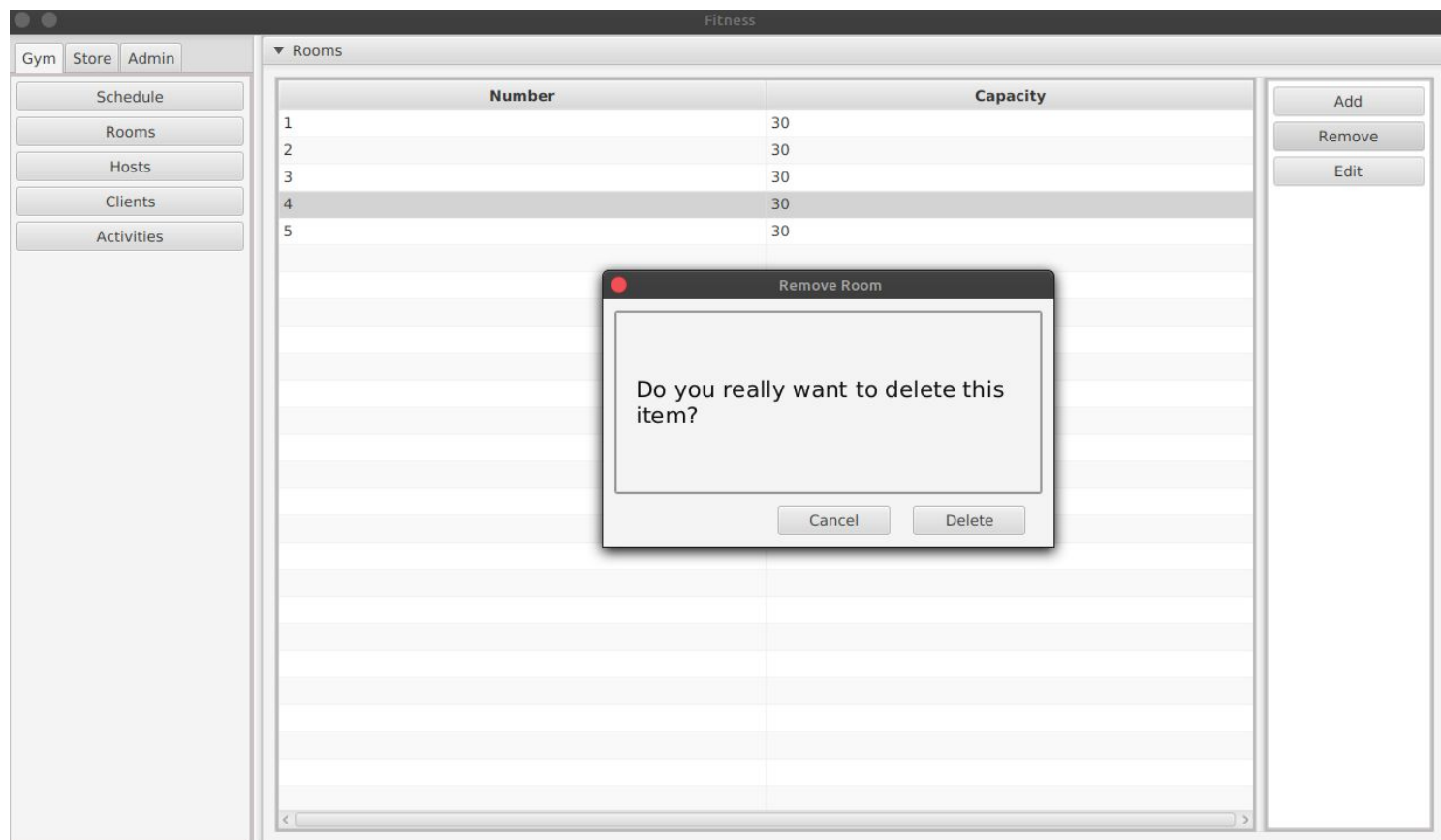


## Edycja sali:

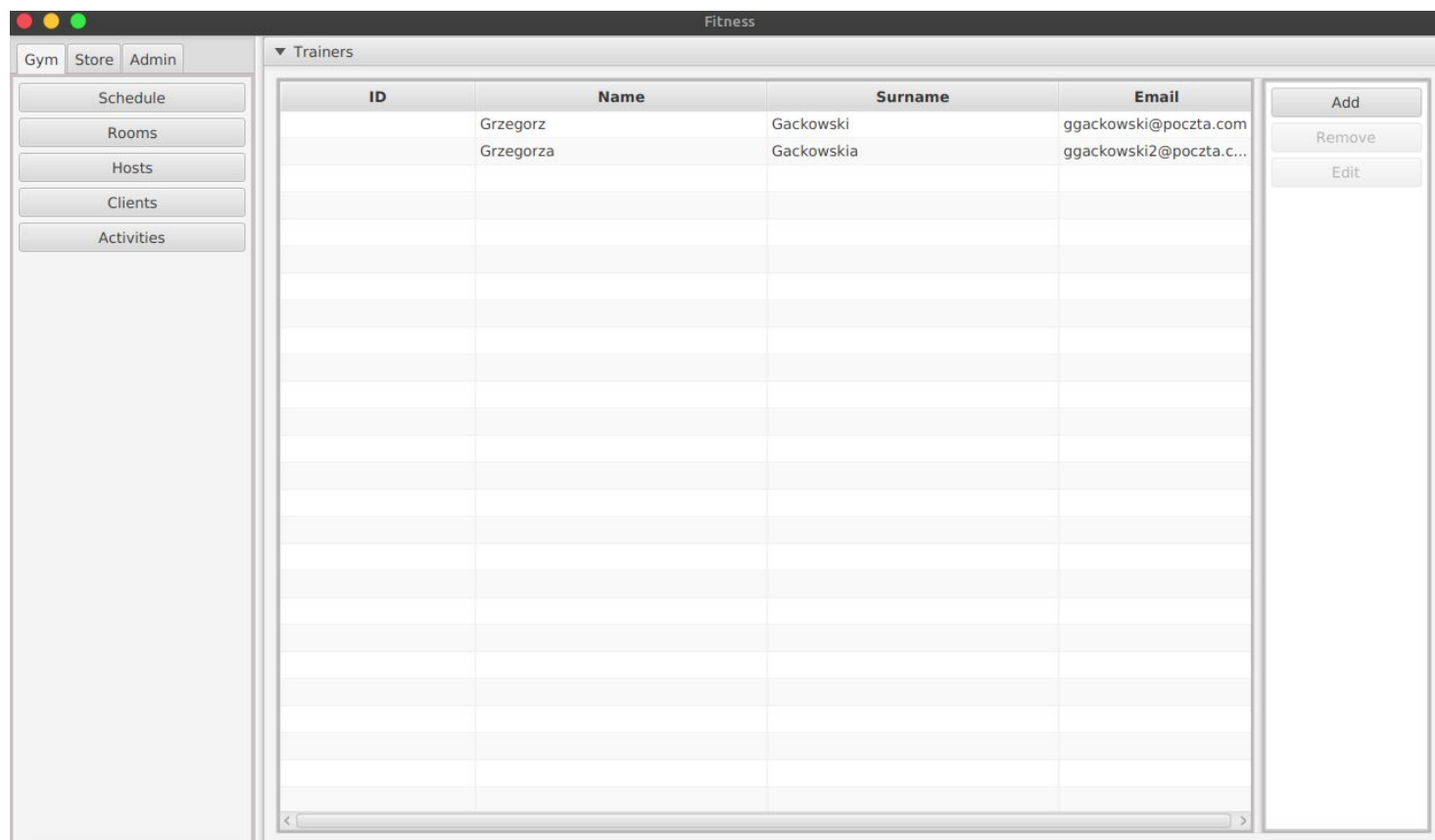




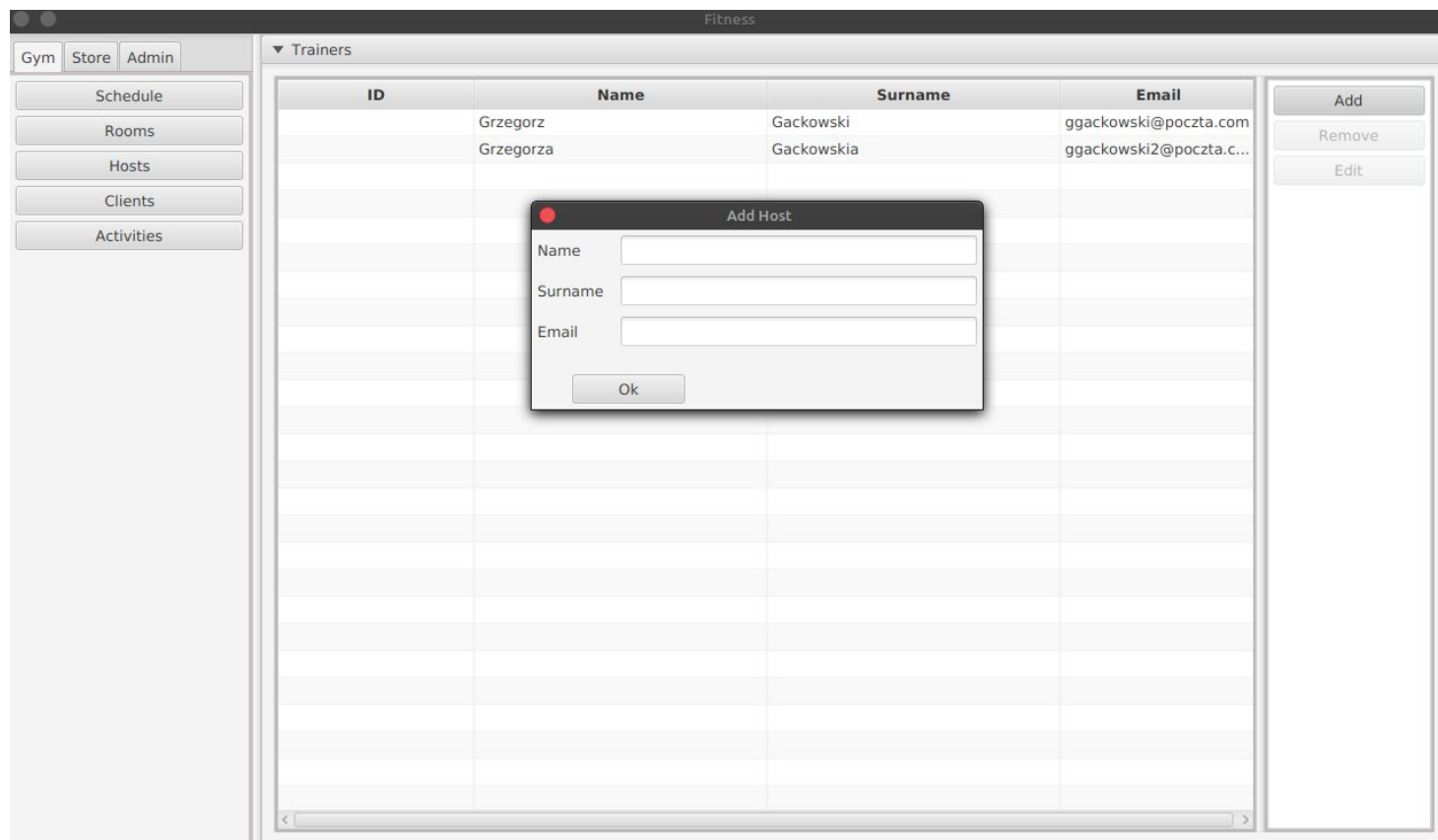
## Usuwanie sali:



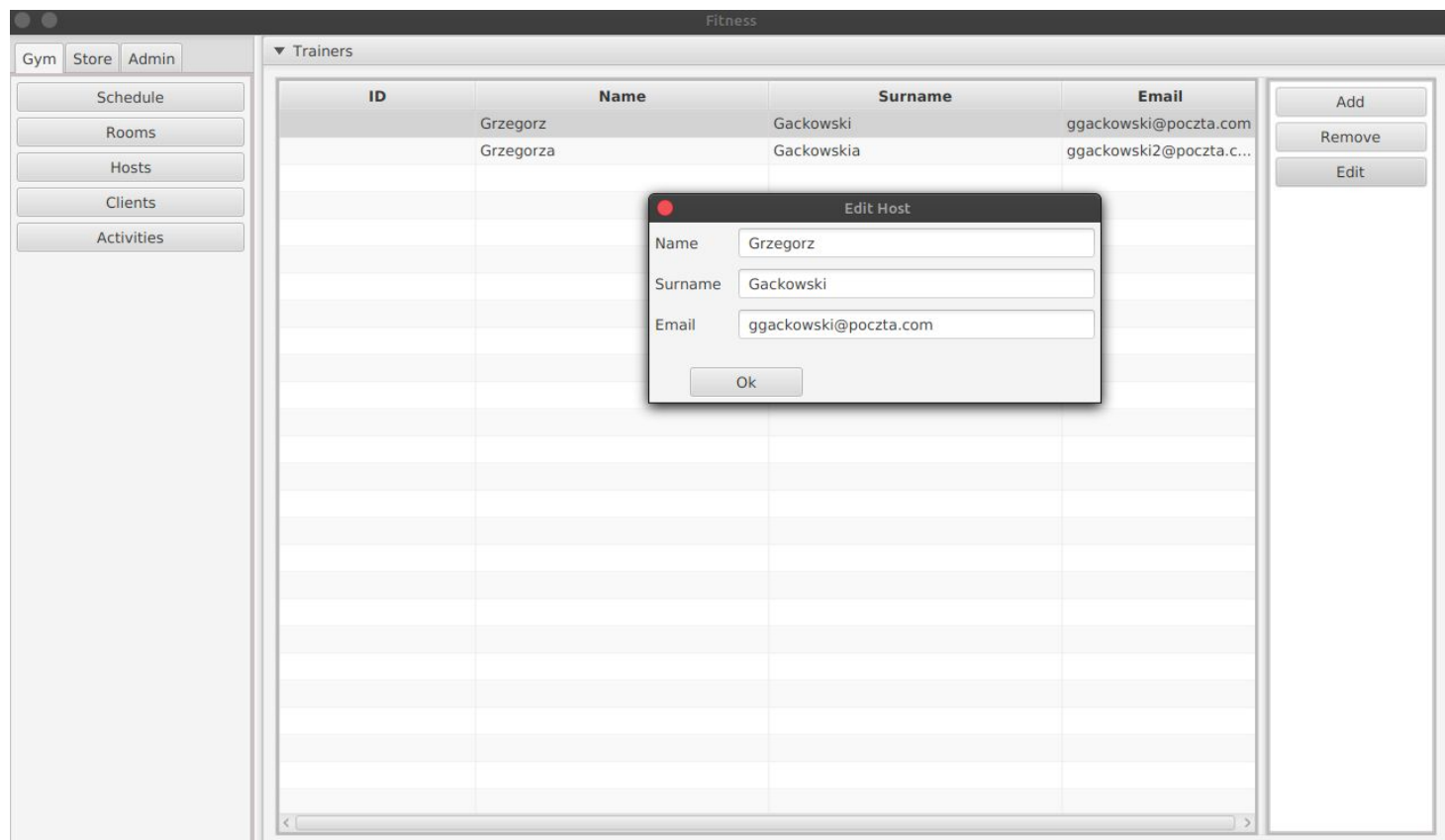
## Lista hostów:



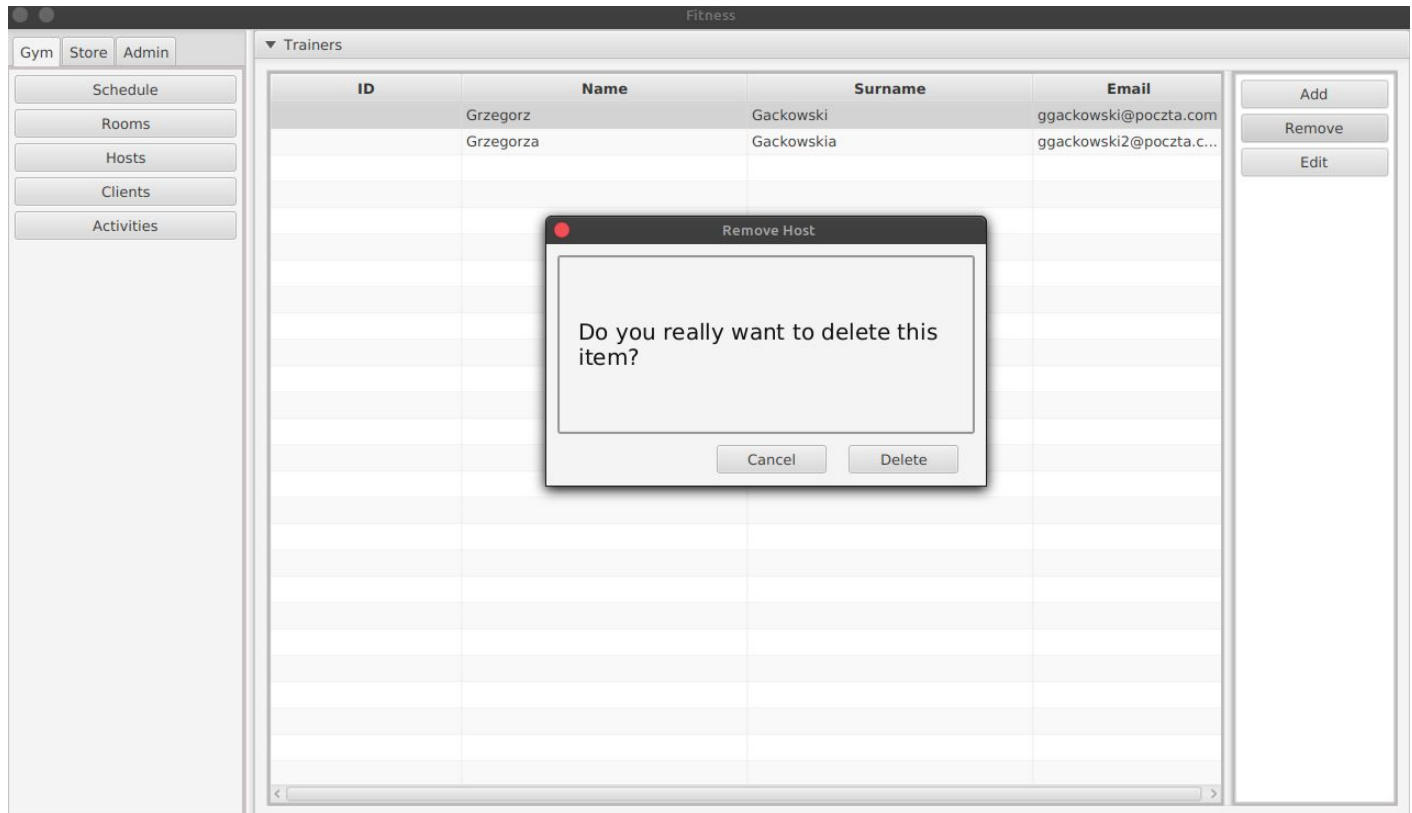
## Dodawanie hosta:



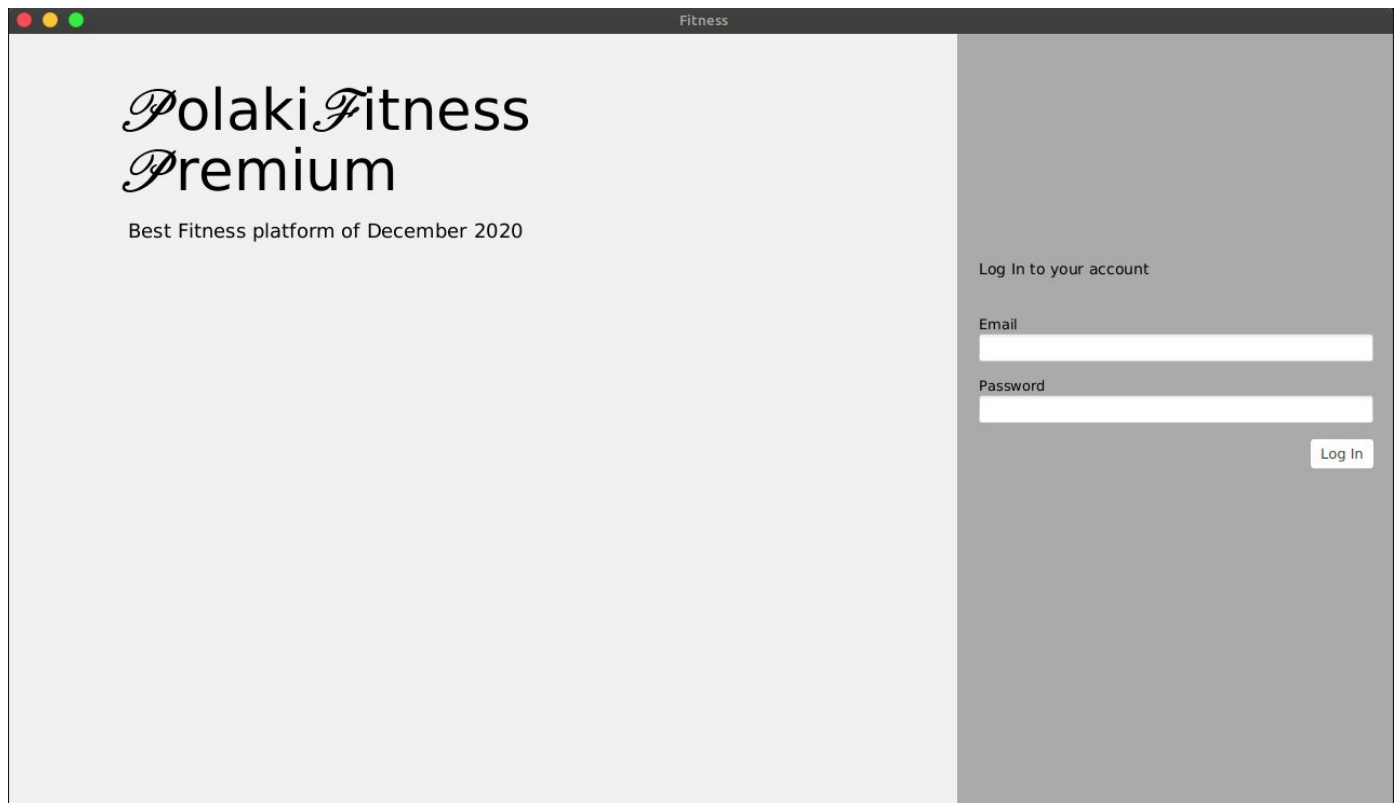
## Edycja hosta:



## Usuwanie hosta



## Ekran logowania:



## Wykorzystane wzorce projektowe

### 1. MVC

W aplikacji zastosowaliśmy wyraźny podział na Model (klasy reprezentujące obiekty biznesowe i zależności pomiędzy nimi), View (definicje widoków zapisane w plikach .fxml) oraz Controller (klasy zarządzające widokami, przesyłające informacje między widokiem a modelem).

Przykładem takiego podziału są klasy Client (Model), plik clients.fxml (View) oraz klasa ClientsController (Controller).

### 2. Wstrzykiwanie zależności

Klasa DataManager związana z persystencją została zbindowana z interfejsem IDataManager. Do każdej klasy, która w swoim konstruktorze potrzebuje obiektu związanego z IDataManager jest wstrzykiwana powiązana z tym interfejsem klasa, czyli w tym momencie DataManager. Dzięki temu w każdym momencie możemy zmienić klasę, która obsługuje persystencję zmieniając jedynie binding interfejsu bez konieczności dokonywania zmian w klasach, które używają danej klasy do obsługi persystencji. Do realizacji tego wzorca wykorzystujemy framework Guice.

```
public class DataModule extends AbstractModule{
    @Override
    protected void configure() { bind(IDataManager.class).to(DataManager.class); }
}
```

```
Injector injector = Guice.createInjector(new DataModule());
dataInitiator = injector.getInstance(DataInitiator.class);
```

```
@Inject
public DataInitiator(IDataManager dataManager) {
    this.dataManager = dataManager;
}
```

```
@Inject
public ActivityManager (IDataManager dataManager) {
    this.dataManager = dataManager;
    instance = this;
}
```

### 3. Singleton

Klasa odpowiadająca za persystencję została oznaczona jako @Singleton i za każdym razem gdy ma zostać gdzieś wstrzyknięty obiekt tej klasy przy

pomocy tego samego Injectora to jest wstrzykiwana zawsze ta sama instancja klasy odpowiadającej za persystencję.

```
@Singleton  
public class DataManager implements IDataManager {
```

- stara wersja

```
@Singleton  
public class HibernateManager implements IDataManager {
```

#### 4. DAO

Każdy element z modelu ma swoją klasę odpowiadającą za dostęp do danych. (1 obiekt DAO (w projekcie nazywany Manager) odpowiada jednej tabeli w bazie.

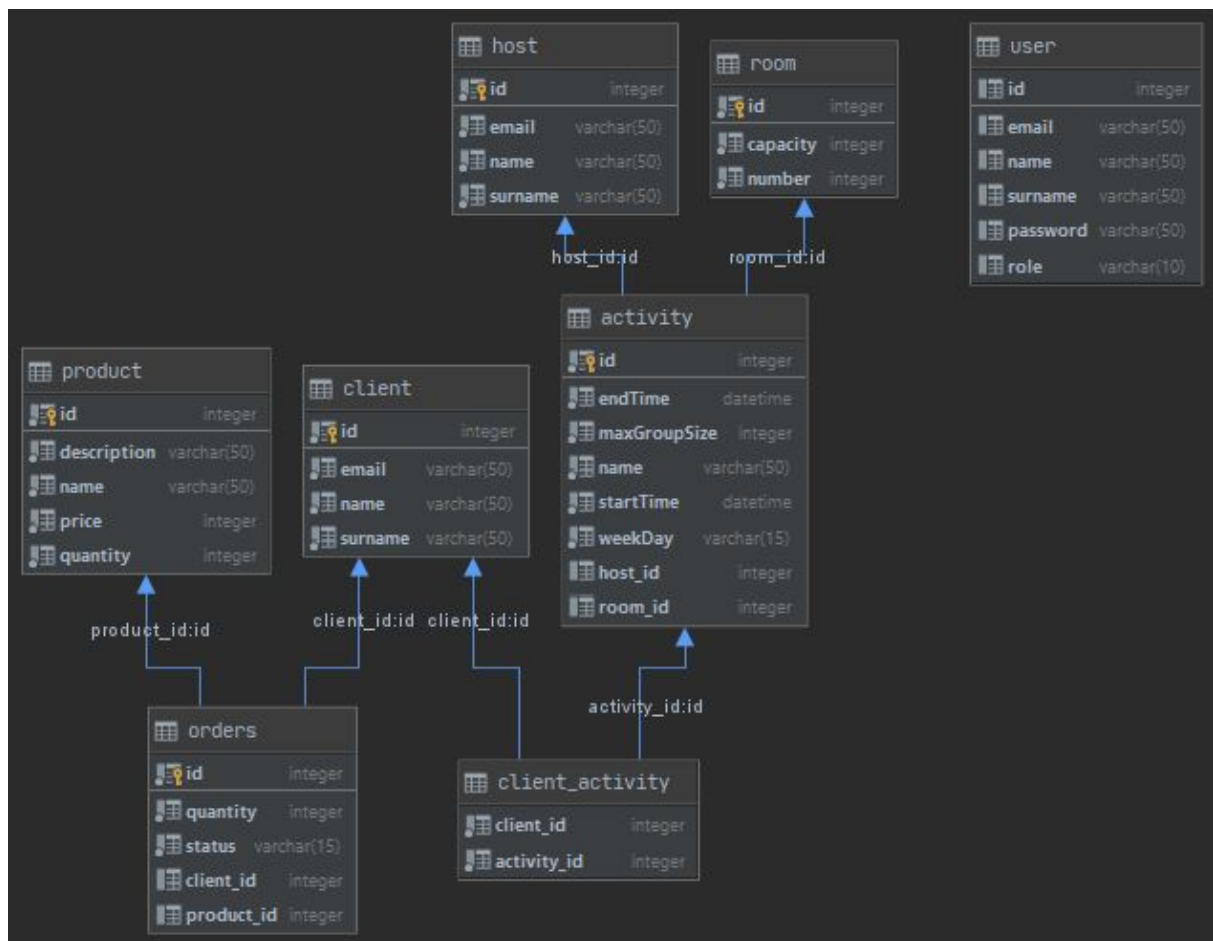


- ActivityManager
- ClientManager
- HostManager
- LoginManager
- RoomManager
- UserManager

#### 5. Data Mapper

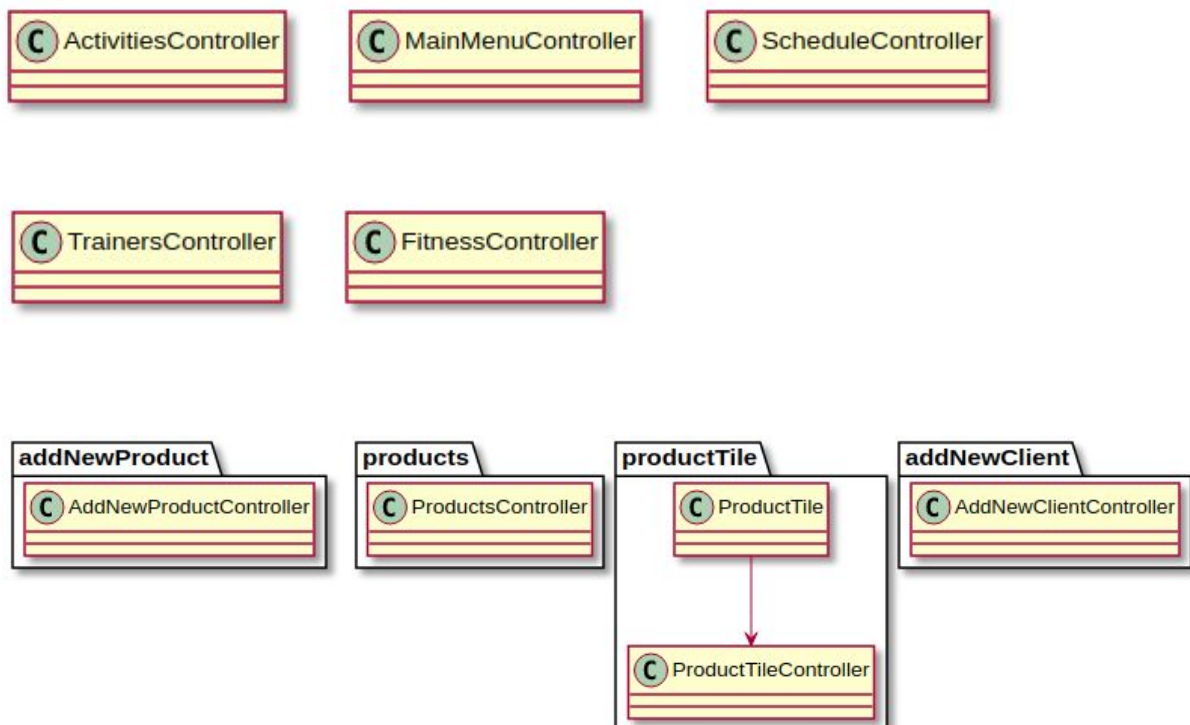
HibernateManager implementujący IDataManager implementuje wszystkie metody dostępu do bazy (pobieranie danych z bazy, usuwanie danych z bazy, aktualizacja danych w bazie). Z tych metod korzystają odpowiednie Managery (np. ActivityManager, który nie ma pojęcia o tym jak jest implementowany bezpośredni dostęp do bazy ale przez IDataManagera ma wszystkie dostarczone metody do operacji na danych).

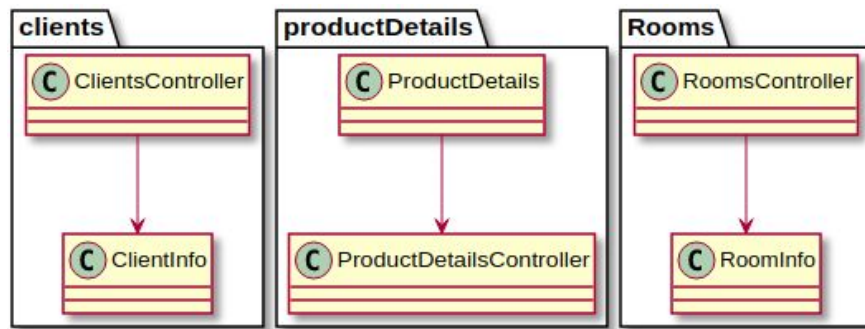
## Diagram bazy



## Diagramy klas

### 1. UI





## 2. Model

