Programowanie komponentowe

Sprawozdanie z pracy projektowej

Kalendarz

Patryk	Lisik	210254
Dominika	Wójcik	210355

Infromatyka, sem IV
2017/2018

Stworzona aplikacja jest kalendarzem przechowującym, wydarzenia, kontakty oraz powiadomienia. Pozwala utrwalać oraz pobierać dane za pomocą formatu XML, serializacji binarnej Javy oraz bazy danych MySQL. Użytkownikowi zostaje udostępniony interfejs graficzny zaprezentowany w podpunktach 3.2.1 - 3.2.3.

1 Najważniejsze klasy warstwy danych projektu

Warstwa danych umożliwia przechowywanie danych o osobach wydarzeniach i powiadomieniach. Osoby są reprezentowane przez klasę Person posiadające dwa pola imię i nazwisko będące stringami. Wydarzenie jest reprezentowane poprzez klasę Event, na którą składają się opis, data początku, końca oraz lista powiadomień. Powiadomienie jest reprezentowane przez klasę Notafication, na którą składają się opis powiadomienia oraz jego data. Warstwa danych przechowuje wyżej wymienione obiekty.

Stworzono dwie implementacje warstwy danych bez połączenia, oraz z połączaniem do bazy danych. Druga z nich rozszerza pierwszą dodając metody pozwalające zapisać zmiany do bazy danych lub wczytać stan z bazy. Połączanie z bazą danych zostało zaimplementowane na zasadzie migawki. Gdy tworzona jest nowa instancja klasy wczytywane są dane z bazy. W tracie wykonania programu w buforze zapisywane są kwerendy zmieniające stan bazy do tożsamego z tym zapisanym w pamięci.

Package dataLayer

Interface Summary

Interface	Description
DataBaseService	Generalisation of database usage
DataService	The Interface DataService.

Class Summary

Class	Description
DataContext	Class that contains data for data service
DataServiceNoSQL	Data service implementation of DataService without SQL queries Class which contains data about events and persons.
DataServiceSQL	Data service which has same functionality like DataServiceNoSQL version, but with every action buffers query to data base use @method syncWithDatabase to save changes to data base.
Event	Event is class which represents event in calendar and contains information about Persons whose are associated with event Remainders date and descriptions By default every event has two remainders one day before and on start
Event.Notification	notification stores date when calendar should notify user about event There is no reason to store duplicated remainders date
Person	Class that represents person .

Rysunek 1: Dokumentacja package'u dataLayer

```
public class DataServiceNoSQL
extends java.lang.Object
implements DataService
```

Data service implementation of DataService without SQL queries Class which contains data about events and persons.

Author:

plisik

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor and Description
DataServiceNoSQL()

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	createEvent(Event ev) Creates the event.
void	createPerson(Person p) Creates the person.
void	deleteEvent(Event ev) Delete event.
void	deleteEvent(int id) Delete event.
void	deletePerson(int id) Delete person.
void	deletePerson(Person p) Delete person.
java.util.HashMap<java.lang.Integer,Event>	getAllEvents() Gets the all events.
java.util.HashMap<java.lang.Integer,Person>	getAllPersons() Gets the all persons.
DataContext	getDataContext() Gets the data context.
Event	getEvent(int id) Gets the event.
Person	getPerson(int id) Gets the person.
void	updateEvent(int id, Event ev) Update event.
void	updatePerson(int id, Person p) Update person.

Rysunek 2: Dokumentacja klasy DataServiceNoSQL

Constructor Summary

Constructors

Constructor and Description
`DataServiceSQL()` Constructor tries to establish connection with data base.

Method Summary

All Methods **Instance Methods** **Concrete Methods**

Modifier and Type	Method and Description
`void`	`addNotification(Event.Notification n)` Adds the notification.
`void`	`addNotificationToEvent(int notificationId, int eventId)` Create entry that connects event and notification in database
`void`	`addPersonToEvent(int personId, int eventId)` Create database entry with person and event
`void`	`createEvent(Event ev)` Add event to database and local structure
`void`	`createPerson(Person p)` Add person to database and local structure
`void`	`deleteEvent(Event ev)` Delete event from database and local structure
`void`	`deleteEvent(int id)` Delete event with particular id from database and local structure
`void`	`deletePerson(int id)` Delete person with particular id from database and local structure
`void`	`deletePerson(Person p)` Delete person from database and local structure
`void`	`delteNotificationsOfEvent(Event ev)` Notifications of particular event will be removed from database
`void`	`delteNotificationsOfEvent(int eventId)` Notifications of particular event will be removed from database
`void`	`loadFromDatabase()` Delete current data and load from database
`void`	`saveToDatabase()` Save changes to database
`void`	`updateEvent(int id, Event ev)` Update event.
`void`	`updatePerson(int id, Person p)` Update person.

Rysunek 3: Dokumentacja klasy DataServiceSQL

Constructor Summary

Constructors

Constructor and Description
`Person()` Instantiates a new person.
`Person(java.lang.String _name, java.lang.String _surname)` Instantiates a new person.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`boolean`	`equals(Person p)` Equals.
`java.lang.String`	`getName()` Gets the name.
`java.lang.String`	`getSurname()` Gets the surname.
`void`	`setName(java.lang.String name)` Sets the name.
`void`	`setSurName(java.lang.String surName)` Sets the sur name.
`java.lang.String`	`toString()`

Rysunek 4: Dokumentacja klasy Person

Class Event

java.lang.Object
dataLayer.Event

All Implemented Interfaces:
java.io.Serializable

```
public class Event
extends java.lang.Object
implements java.io.Serializable
```

Event is class which represents event in calendar and contains information about Persons whose are associated with event Remainders date and descriptions By default every event has two remainders one day before and on start

Author:
plisik

See Also:
Serialized Form

Nested Class Summary

Nested Classes

Modifier and Type	Class and Description
static class	Event.Notification notification stores date when calendar should notify user about event There is no reason to store duplicated remainders date

Constructor Summary

Constructors

Constructor and Description
Event(java.lang.String name, java.util.Date start, java.util.Date end)
Event(java.lang.String name, java.util.Date start, java.util.Date end, java.util.Set<java.lang.Integer> associatedPersons)

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	addAssociatedPerson(java.lang.Integer personId) Add person by id
void	addNotification(java.util.Date date, java.lang.String descripton) Create notification from date and description
java.util.Set<java.lang.Integer>	getAssociatedPersons()
java.util.Date	getEnd()
java.lang.String	getName()
java.util.Map<java.lang.Long,Event.Notification>	getNotifications()
java.util.Date	getStart()
void	setAssociatedPersons(java.util.Set<java.lang.Integer> associatedPersons)
void	setEnd(java.util.Date end)
void	setName(java.lang.String name)
void	setNotifications(java.util.Map<java.lang.Long,Event.Notification> remainders)
void	setStart(java.util.Date start)
java.lang.String	toString()

Rysunek 5: Dokumentacja klasy Event

2 Najważniejsze klasy warstwy logiki

Warstwa logiki umożliwia manipulowanie warstwą danych. Poza tworzeniem obiektów warstwy danych warstwa logiki umożliwia:

1. znalezienie wydarzeń między datami
2. znalezienie wydarzeń w danym dniu
3. posortowanie wydarzeń po dacie
4. posortowanie wydarzeń po ilości uczestniczących osób
5. zapis i odczyt z XML i serializacji binarnej
6. zapis do formatu odt(OpenOffice)
7. informowanie o zmianie stanu kolekcji

Do seralizowania danych do formatu XML wykorzystano bibliotekę XStream.

Zapis i odczyt z formatów standardowych zrealizowano za pomocą wzorca projektowego strategi. Stworzone zostały interface'y Importer i Saver które uogólniają zapis i odczyt z dowolnego formatu. Na ich podstawie powstały konkretne klasy jak XMLSaver,XMLImporter,ODTSaver itd.

Zaimplementowane zostało automatyczne uruchamianie powiadomień. Odpowiada za to klasa EventNotifiactionPublisher posiadająca dwa wewnętrzne interfac'y:

- NotificationReciver – uogólniający dowolnego odbiorcę powiadomień, do którego o odpowiednim czasie(zdefiniowanym przez samo powiadomienie) zostanie wysłana teść powiadomienia(poprzez wywołanie odpowiedniej metody).
- NotifiactionSource – uogólnienie źródła powiadomień. oczekuje się że wywołana odpowiednią metodę jeśli lista powiadomień zostanie zmieniona.

Sama czynność wywoływania metody w odpowiednim czasie została zrealizowana za pomocą kalsy TimerTask.

Interface Importer

All Known Implementing Classes:
BinaryImporter, XMLImporter

---

public interface **Importer**

Interface that abstract out importing from different formats and APIs Corresponding Importer and Saver implementation should be compatible

Author:
plisik

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
DataService	importData(java.lang.String fileName)

Rysunek 6: Dokumentacja interfac'u Importer

All Known Implementing Classes:
BinarySaver, OpenOfficeSaver, XMLSaver

---

public interface **Saver**

Interface to abstract out serialisation to different formats and exporting to different APIs Corresponding Importer and Saver implementation should be compatible

Author:
plisik

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
void	save(java.lang.String filename, DataService data)

Rysunek 7: Dokumentacja interfac'u Saver

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	addEventNotificationPublisher(EventNotifiactionPublisher enp)
void	addNotification(int EventId, java.util.Date date, java.lang.String description) Adds the notification.
void	addPersonToEvent(int EventId, int PersonId) Adds the person to event.
void	createEvent(java.lang.String name, java.util.Date start, java.util.Date end) Creates the event.
void	createPerson(java.lang.String name, java.lang.String surname) Creates the person.
void	deleteEvent(Event ev) Delete event.
void	deleteEvent(int id) Delete event.
void	DeleteEventsBetweenDates(java.util.Date start, java.util.Date end) Delete events in range
void	deletePerson(int id) Delete person.
void	deletePerson(Person p) Delete person.
java.util.List<Event>	EventBetweenDate(java.util.Date start, java.util.Date end) Events between dates
java.util.List<Event>	EventsByDate() Events by date.
java.util.List<Event>	EventsByNumberOfParticipants() List of events sorted by number of participants
java.util.List<Event>	EventsOn(java.util.Date start) Events on particular day
java.util.Set<java.lang.Integer>	getAllAssociatedPersons(int eventId) Gets the all associated persons.
java.util.Map<java.lang.Integer,Event>	getAllEvents() Gets the all events.
java.util.Map<java.lang.Long,Event.Notification>	getAllNotification(int eventId) Gets the all notification.
java.util.List<Event.Notification>	getAllNotifications()
java.util.Map<java.lang.Integer,Person>	getAllPersons() Gets the all persons.
DataService	getDataService() Gets the data service.
Event	getEvent(int id) Gets the event.
Person	getPerson(int id) Gets the person.
void	importFromBianry(java.lang.String fileName) Override current by state from binary file
void	importFromXML(java.lang.String fileName) Override current by state from XML file
void	removeNotification(int EventId, java.lang.Long NotificationId) Removes the notification.
void	removePersonFromEvent(int EventId, int PersonId) Removes the person from event.
void	saveToBianry(java.lang.String fileName) Save current sate to binary file
void	saveToODT(java.lang.String fileName) Export to OpenOffice file
void	saveToXML(java.lang.String fileName) Save current sate to XML file
void	updateEvent(int id, Event ev) Update event.
void	updatePerson(int id, Person p) Update person.

Rysunek 8: Dokumentacja klasy LogicLayerImpl

Constructor Summary

Constructors

Constructor and Description
`LogicLayerSQLImpl(DataService data)`

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`void`	`loadFromDatabase()` inject data from data base
`void`	`saveToDatabase()` save current state to dataBase

Rysunek 9: Dokumentacja klasy LogicLayerSQLImpl

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`static LogicLayerImpl`	`getLogicLayerNoSQL()` Logic layer without database
`static LogicLayerSQLImpl`	`getLogicLayerSQL()` Logic layer with database

Rysunek 10: Dokumentacja klasy LogicLayerFactory

Nested Class Summary

Nested Classes

Modifier and Type	Class and Description
static interface	EventNotifiactionPublisher.NotifiactionSource Notification source is supposed to invoke update method when notification list is changed
static interface	EventNotifiactionPublisher.NotificationReciver

Constructor Summary

Constructors

Constructor and Description
EventNotifiactionPublisher(EventNotifiactionPublisher.NotifiactionSource source)

Method Summary

All Methods **Instance Methods** **Concrete Methods**

Modifier and Type	Method and Description
void	addReciver(EventNotifiactionPublisher.NotificationReciver reciver) Add notification reviver
void	update() Delete all notifications and reschedule all

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

EventNotifiactionPublisher

```
public EventNotifiactionPublisher(EventNotifiactionPublisher.NotifiactionSource source)
```

Rysunek 11: Dokumentacja klasy EventNotificationPublisher

3 Warstwa interfejsu

3.1 Najważniejsze klasy warstwy interfejsu

Główną klasą odpowiedzialną za uruchomienie wszystkich mechanizmów związanych z widokiem aplikacji jest klasa MainWindow. Podczas uruchomienia aplikacji powstaje obiekt klasy Calendar odpowiedzialny za główny widok okna oraz utworzenie słuchaczy, czyli obiektów klasy implementującej interfejs ActionListener. Pozostałe klasy takie jak ContactsView, DayView, MonthView i inne implementują pozostałe elementy kalendarza. W warstwie prezentacji znaj-

duje się także klasa StateContainer odpowiedzialna za zarządzanie generowanymi podczas działania programu zdarzeniami.

./screen/GUI/calendar.png

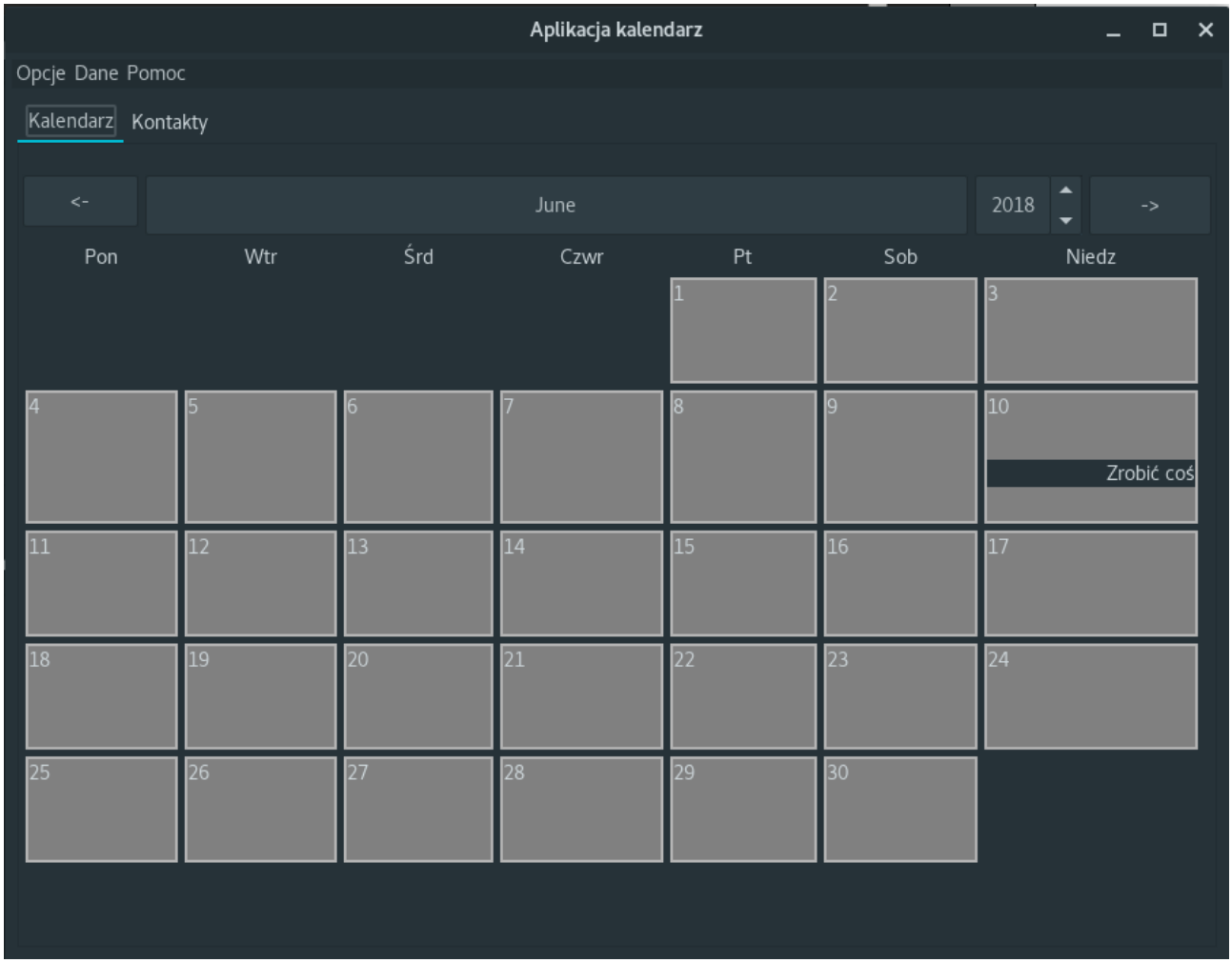
Rysunek 12: Dokumentacja klasy calendar

./screen/GUI/stateContainer.png

Rysunek 13: Dokumentacja klasy stateContainer

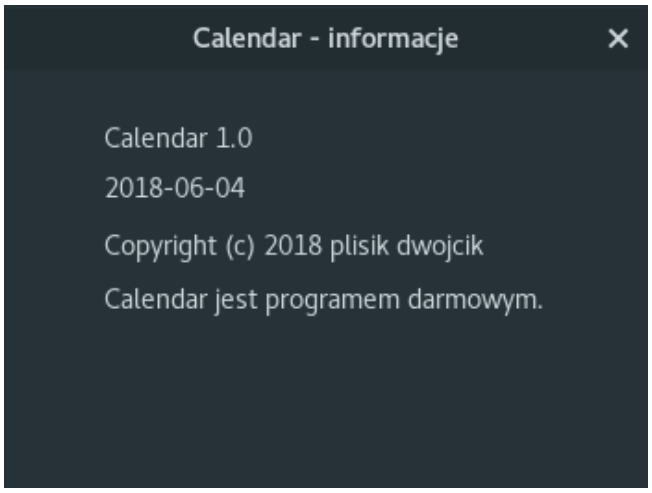
3.2 Zrzuty ekranu najważniejszych widoków działającej aplikacji

3.2.1 Główne okno kalendarza "organizera"



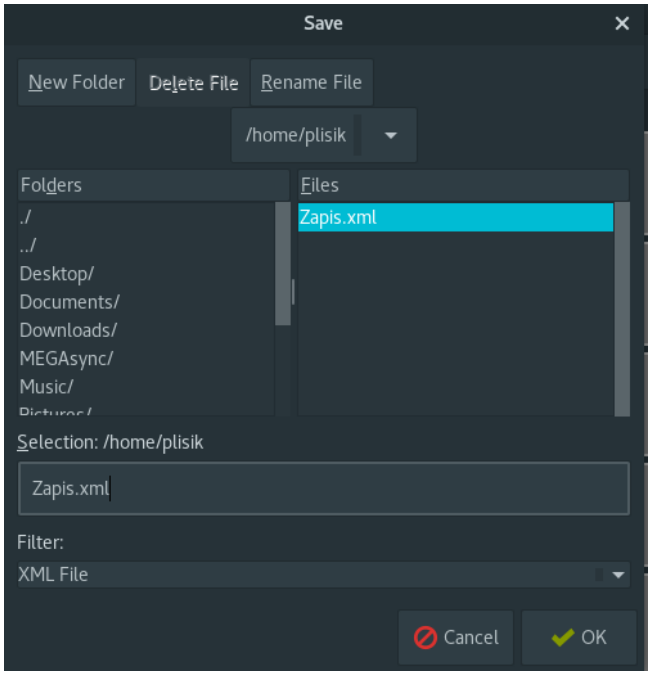
Rysunek 14: Główne okono kalendarze w sytstemi linux

3.2.2 Okienko "O programie"

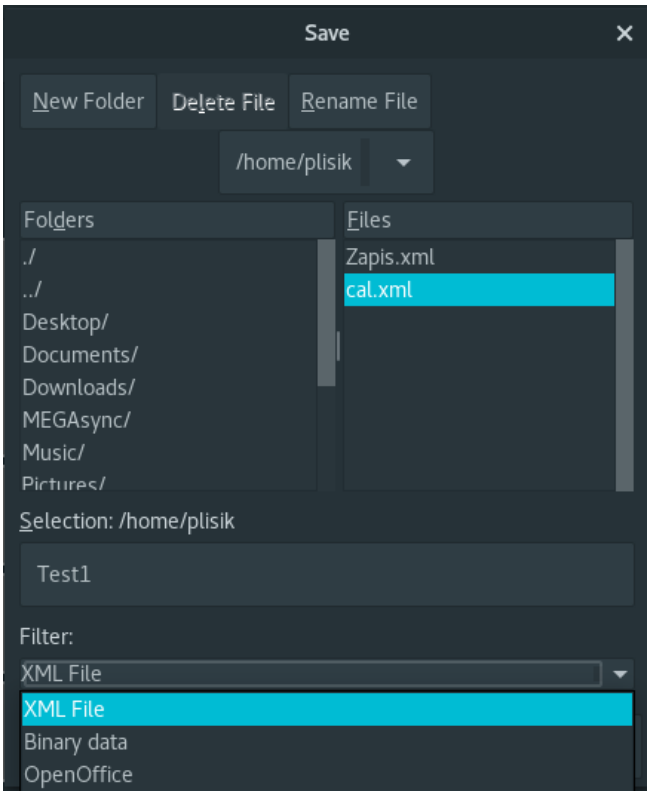


Rysunek 15: Okno pokazujące informacje o programie

3.2.3 Okienka dialogowe zapisujące/odczytujące dane i ustawienia do/z bazy-/pliku XML



Rysunek 16: Okno zapisu do XML



Rysunek 17: Wybór formatu zapisu w oknie zapisu

3.3 Jasny i czytelny opis klawiszy funkcyjnych i przycisków obsługujących aplikację

4 Postać bazy danych z zapisanymi informacjami o zdarzeniach

W projekcie zastosowano bazę danych MySQl w wersji 8.0.11. Wybór ten podyktowany był znacznie prostszą konfiguracją w systemie linux(wystarczy jedna komenda w terminalu).

```
DROP DATABASE IF EXISTS calendar_data;
CREATE DATABASE calendar_data;

use calendar_data;


CREATE TABLE events(
    id int,
    name varchar(30),
    start DATETIME,
    end DATETIME
);


ALTER TABLE events
```

```sql
ADD PRIMARY KEY (id);

CREATE TABLE persons(
    id int,
    name varchar(20),
    surname varchar(20)
);

ALTER TABLE persons
ADD PRIMARY KEY (id);

CREATE TABLE notifications(
    id int,
    description varchar(40),
    notify_date DATETIME
);

ALTER TABLE notifications
ADD PRIMARY KEY (id);

CREATE TABLE event_person(
    person_id int,
    event_id int
);

ALTER TABLE event_person
ADD FOREIGN KEY (person_id) REFERENCES persons(id);
ALTER TABLE event_person
ADD FOREIGN KEY (event_id) REFERENCES events(id);

CREATE TABLE notifications_events(
    event_id int,
    notification_id int
);


DESCRIBE persons;
DESCRIBE event_person;
DESCRIBE notifications_events;
DESCRIBE notifications;
```

Listing 1: Skrypt tworzący bazę danych

```
mysql> SELECT * FROM events;
+----+------+---------------------+---------------------+
| id | name | start               | end                 |
+----+------+---------------------+---------------------+
|  1 | test | 2018-06-13 22:00:00 | 2018-06-16 06:57:18 |
+----+------+---------------------+---------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM persons;
+----+--------+---------+
| id | name   | surname |
+----+--------+---------+
|  1 | patryk | lisik   |
+----+--------+---------+
1 row in set (0.00 sec)
```

```

mysql> SELECT * FROM notifications;
+----+-----------------------------+---------------------+
| id | description                 | notify_date         |
+----+-----------------------------+---------------------+
|  0 | is starting now             | 2018-06-13 22:00:00 |
|  1 | is going to start in one day | 2018-06-12 22:00:00 |
+----+-----------------------------+---------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM notifications_events;
+----------+-----------------+
| event_id | notification_id |
+----------+-----------------+
|        0 |               1 |
|        1 |               1 |
+----------+-----------------+
2 rows in set (0.00 sec)
```

Listing 2: Listing z stanu bazy danych po wykonaniu kilku operacji

5 Podsumowanie i ewentualne uwagi grupy nt. projektu.

Programistyczna strona projektu, poza kilkoma nieporozumieniami związanymi z wykorzystaniem klas zaimplementowanych przez drugą osobę, nie sprawiła większych problemów. Popełniliśmy natomiast kilka błędów związanych z używanymi narzędziami:

- Zbyt późno zaczęliśmy używać Mavan'a do zarządzania zależnościami, przez co często traciliśmy czas na doinstalowywanie bibliotek i edycję pliku "BuildPatch".
- Brak tworzenia testów automatycznych i ciągłej integracji poskutkował kilkoma regresami i trudnymi do zlokalizowania błędami wynikającymi z niewielkich zmian.