НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра

Обчислювальної техніки

КУРСОВА РОБОТА
з «ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

на тему: «Система організації вуличних змагань «WorkOUT». Робоче місце гостя »

Студента 2 курсу групи ІО-32
напряму підготовки
6.050102 «Комп'ютерна інженерія»

Попенко Руслан Леонідович

Керівник
Болдак Андрій Олександрович
(прізвище та ініціали)

Доцент кафедри ОТ
(посада, вчене звання, науковий ступінь)

Національна шкала_____
Кількість балів: _____
Оцінка: ECTS _____

м. Київ - 2015 рік

РОЗДІЛ 1
ЗАПИТИ ЗАЦІКАВЛЕНИХ ОСІБ

Введення

У цьому документі описані запити зацікавлених осіб по відношенню до розроблюваної системи «Організація вуличних змагань WorkOut», в якості яких виступають: організатори цих змагань, судді, учасники та гості(глядачі).

1.1. Мета

Метою документа є визначення основних вимог щодо організації даних спортивних змагань, а саме таких аспектів, як забезпечення інформацією усіх зацікавлених осіб, їх взаємодії за посередництвом системи.

1.2. Контекст

Перелік вимог, перерахованих у цьому документі, є основою технічного завдання для розробки системи для супроводу вуличних спортивних змагань, виставлення балів (оцінок) суддями, забезпечення інформацією учасників та гостей.

1.3. Короткий огляд продукту

Систему «Організація вуличних змагань WorkOut» можна умовно розділити на дві частини програмного забезпечення:

Сайт, який використовують зацікавлені особи в якості інтерфейсу доступу до усіх даних, що пов’язані із змаганнями, інструменту для роботи з ними;

База даних на головному сервері, що відображатиметься на сайті у вигляді турнірної таблиці.

1.4. Ділові правила і приписи

1.4.1. Призначення системи організації вуличних змагань

Система призначена для забезпечення взаємодії між базою даних та сайтом змагань, а також встановлення формальних правил доступу до даних та можливостей роботи з ними для потенційних користувачів системою.

Така організація зберігання даних про учасників, місце, час змагань та іншої інформації виключає можливість втрати інформації або порушення її конфіденційності.

Також, використовуючи вищезазначену систему, можна отримати інформацію щодо будь-якого учасника та подальшу його роль у змаганнях.

Отже, завдання системи «Організація вуличних змагань WorkOut» - заміна її потенційного паперового аналогу більш сучасною, доступною і зручною у використанні системою.

1.4.2. Політика взаємодії із зацікавленими особами

При використанні системи «Організація вуличних змагань WorkOut» всі зацікавлені особі мають можливість використовувати програмне забезпечення системи відповідно до їх рівня доступу. Умови доступу до цієї системи встановлює політика конфіденційності.

Організатори мають повний доступ до системи, можуть вносити корективи щодо учасників, місця і часу проведення змагань, нагородження переможців і т.д., за виключенням оцінювання учасників.

Судді мають доступ до всієї інформації про змагання, але вносити зміни мають право лише щодо оцінок.

Учасники реєструються у змаганнях, мають повний доступ до загальної інформації та до власного кабінету.

Гості мають доступ лише до загальної інформації.

1.5. Сценарії

1.5.1. Сценарій реєстрації у змаганнях

Назва: сценарій реєстрації у змаганнях.

Учасники: гість, організатор.

Передумови: гість не зареєстрований у системі.

Результат: зареєстровано нового учасника змагань.

Сценарій:

1. Гість заповнює анкету своїми даними на сайті. Для підтвердження особи він прикріплює фотографію або скан будь-якого документу, що засвідчує особу.
2. Гість надсилає анкету організаторам.
3. Автоматично створюється неактивний профіль учасника.

4. Організатор отримує інформацію.

5. Організатор надсилає на адресу електронної пошти гостя запит про підтвердження реєстрації.

6. Гість підтверджує факт реєстрації у змаганнях.

7. Організатор відкриває базу даних системи та вносить відповідні дані про учасника.

8. Організатор створює для учасника особистий електронний кабінет.

9. Учасник отримує персональний ідентифікаційний номер та пароль для подальшого доступу до особистого кабінету.

10. Профіль учасника автоматично активується.

11. По завершенню роботи організатор закриває базу даних.

12. Організатор перевіряє наявність змін у турнірній таблиці на сайті.

Виключні ситуації:

1. Гість невірно заповнив поля анкети або залишив їх порожніми.

2. Гість не підтвердив факт реєстрації у змаганнях.

3. Порушено зв’язок між сервером із базою даних та сайтом.

1.5.2. Сценарій доступу до особистого кабінету

Назва: сценарій доступу до особистого кабінету.

Учасники: учасник.

Передумови: учасник зареєстрований у змаганнях.

Результат: перевірено поточні дані учасника.

Сценарій:

1. Учасник вводить ідентифікаційний номер та пароль.

2. Учасник отримує доступ до особистого кабінету.

3. Автоматично створюється сеанс роботи.

4. Учасник переглядає інформацію щодо своїх успіхів, а також місця і часу проведення наступного етапу змагань, у якому він бере участь.

5. Учасник виходить із особистого кабінету.

6. Сеанс роботи автоматично завершується.

Виключні ситуації:

Учасник невірно ввів ідентифікаційний номер та пароль або залишив поля порожніми.

1.5.3. Сценарій відмови від участі у змаганнях.

Назва: сценарій відмови від участі у змаганнях.

Учасники: учасник, організатор.

Передумови: учасник зареєстрований у змаганнях.

Результат: учасник усунений від участі.

Сценарій:

1. Учасник заходить до особистого кабінету.

2. Учасник надсилає організатору повідомлення про бажання відмовитись від участі у змаганнях.

3. Учасник виходить із особистого кабінету.

4. Організатор отримує повідомлення.

5. Організатор виключає запис учасника із бази даних.

6. Організатор видаляє особистий кабінет учасника.

7. Організатор перевіряє наявність змін у турнірній таблиці на сайті.

Виключні ситуації:

1. Порушено зв’язок між сервером із базою даних та сайтом.

2. Учасник не вийшов з особистого кабінету.

1.5.4. Сценарій адміністрування ресурсів організаторами

Назва: сценарій адміністрування ресурсів організаторами.

Учасники: організатор.

Передумови: організатор має доступ до адміністрування.

Результат: внесено зміни до даних на сайті.

Сценарій:

1. Організатор вводить логін та пароль для доступу до бази даних.
2. Організатор вносить зміни до бази даних відповідно до повідомлень з боку учасників та судей.
3. Організатор завершує роботу та залишає сервер з базою даних.
4. Організатор вводить шестизначний логін та персональний пароль для доступу до адміністрування на сайті.
5. Автоматично створюється сеанс роботи.
6. Організатор працює з ресурсами сайту відповідно до своїх прав доступу.
7. Організатор закінчує роботу.
8. Сеанс роботи автоматично завершується.

Виключні ситуації:

1. Організатор невірно ввів логін та пароль або залишив поля порожніми.
2. Порушено зв’язок між сервером із базою даних та сайтом.

1.5.5. Сценарій адміністрування ресурсів суддями

Назва: сценарій адміністрування ресурсів суддями.

Учасники: суддя.

Передумови: суддя має частковий доступ до адміністрування.

Результат: внесено зміни до турнірної таблиці.

Сценарій:

1. Суддя вводить шестизначний логін та персональний пароль для отримання часткового доступу до адміністрування на сайті.
2. Автоматично створюється сеанс роботи.

3. Суддя працює з оцінками учасників змагань.

4. Суддя надсилає організатору запит на внесення змін до турнірної таблиці.

5. Суддя закінчує роботу.

6. Сеанс роботи автоматично завершується.

Виключні ситуації:

1. Суддя невірно ввів логін та пароль або залишив поля порожніми.

2. Порушено зв’язок між сайтом та сервером із базою даних.

1.5.6. Сценарій переходу з категорії «учасник» до категорії «організатор».

Назва: сценарій переходу з категорії «учасник» до категорії «організатор».

Учасники: учасник, організатор.

Передумови: учасник бажає організувати власні змагання.

Результат: зареєстровано нового організатора.

Сценарій:

1. Учасник заходить до особистого кабінету.

2. Учасник надсилає організаторам повідомлення, що він бажає стати організатором власних змагань.

3. Учасник виходить з особистого кабінету.

4. Організатор проводить співбесіду з іншими організаторами.

5. Організатор надсилає учасникові підтвердження, персональні логін та пароль для доступу до адміністрування на сайті та дані для доступу до бази даних.

6. Кожному учасникові, який перейшов у категорію організаторів, на перший час надається персональний консультант.

Виключні ситуації:

На співбесіді організатори вирішили відхилити запропоновану кандидатуру.

1.5.7. Сценарій переходу з категорії «учасник» до категорії «суддя».

Назва: сценарій переходу з категорії «учасник» до категорії «суддя».

Учасники: учасник, організатор.

Передумови: учасник бажає бути суддею на змаганнях.

Результат: зареєстровано нового суддю.

Сценарій:

1. Учасник заходить до особистого кабінету.
2. Учасник надсилає організаторам повідомлення, що він бажає стати суддею змагань.
3. Учасник виходить з особистого кабінету.
4. Організатор створює загальне голосування на сайті, у якому можуть брати участь усі зацікавлені особи.
5. Організатор надсилає учасникові підтвердження та персональні логін та пароль для внесення змін до турнірної таблиці.
6. Кожному учасникові, який перейшов у категорію судей, на перший час надається персональний консультант.

Виключні ситуації:

В результаті голосування виявилося, що більшість прости запропонованої кандидатури.

1.6. Функціональність системи

Основні вимоги до функціональності, пред'явлені зацікавленими особами, відносяться до чотирьох категорій :

1. Організатор
2. Суддя
3. Учасник
4. Гість

1.6.1. Структура сайту системи «Організація вуличних змаганьWorkOut»

Сайт системи повинен мати наступні розділи:

1. ***Реєстрація* – розділ, в якому будь-який бажаючий зможе зареєструватися у змаганнях.**
2. ***Вхід до особистого кабінету* – дає можливість переглянути персональну сторінку учасника.**
3. *Турнірна таблиця* – інформація про оцінки кожного учасника та відомість про поточне місце учасника в турнірі.
4. *Керування та оцінювання* – розділ виключно для організаторів та суддів, в ньому можна зареєструвати учасника, внести зміни до турнірної таблиці, опублікувати оголошення щодо проведення змагань.
5. *Гостям турніру* – розділ, в якому гість може забронювати сидяче місце на змаганні.
6. *Оголошення* – інформація про адресу та час проведення змагань.
7. *Новини* – інформація про поточний хід подій на змаганнях.
8. *Контакти* – телефони для довідок організаторів змагання, адреса за яким можна звертатися.
9. *Архіви змагань* –інформація про турніри, які вже відбулися.

1.6.2. Можливості зацікавлених осіб

А)Можливості учасника

Реєструватися для участі у змаганнях

Мати доступ до свого особистого кабінету

Надіслати організаторам запит щодо організації власних змагань

Надіслати організаторам запит щодо переходу у категорію судей

Переглядати новини

Переглядати оголошення

Коментувати новини

Переглядати турнірну таблицю

Переглядати архів попередніх змагань

Переглядати контактну інформацію.

Б)Можливості гостя (глядача)

Переглядати новини

Переглядати оголошення

Переглядати архів попередніх змагань

Переглядати контактну інформацію

Реєструватися для участі у змаганнях

Переглядати турнірну таблицю.

В)Можливості організаторів

Мати персональний доступ до керування сайтом

Реєструвати учасників

Публікувати новини

Публікувати оголошення

Переглядати новини

Переглядати оголошення

Вносити зміни до турнірної таблиці

Переглядати турнірну таблицю

Завантажувати архів змагань

Переглядати архів змагань

Публікувати контактну інформацію

Переглядати контактну інформацію.

Г)Можливості суддів

Мати персональний доступ до редагування турнірної таблиці

Переглядати новини

Переглядати оголошення

Вносити зміни до турнірної таблиці

Переглядати турнірну таблицю

Переглядати архів змагань

Переглядати контактну інформацію.

1.7. Практичність

1.7.1. Стандартизація

Система надає доступ до інформації кожному її користувачеві відповідно до його приналежності до однієї з чотирьох категорій.

1.7.2. Інтерфейс користувача

Інтерфейс сайту системи «Організація вуличних змагань WorkOut» відповідає наступним вимогам:

Зрозумілий і не допускає двозначного тлумачення.

Виконаний з урахуванням ергономічних вимог.

Всі кодовані параметри або елементи, наведені скорочення повинні мати розшифрування або вікно-підказку, що буде з'являтися після наведення курсору на елемент або після натискання спеціальної клавіші.

1.8. Надійність.

Протягом терміну зберігання архівів змагань повинна бути забезпечена їх цілісність.

Протягом терміну проведення одного турніру повинна бути забезпечена недоторканність та достовірність особистих сторінок учасників поточних змагань.

Для забезпечення збереження та цілісності організаторами буде використовуватися метод резервного копіювання.

Для забезпечення незмінності та достовірності організаторами буде використовуватися комплекс технологічних і адміністративних процедур, що перешкоджають випадковій або навмисній зміні збережених даних із бази.

Також повинна бути забезпечена конфіденційність персональної інформації. Надання доступу до персональних даних організатори здійснюють у відповідності з правами доступу відвідувача.

РОЗДІЛ 2
РОЗРОБКА ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Загальна схема прецедентів

Загальна схема прецедентів для ролі гостя показує можливі послідовності дій даної категорії та встановлює зв'язок дій між різноманітними категоріями. Схема прецедентів представлена на рис. 2.1.

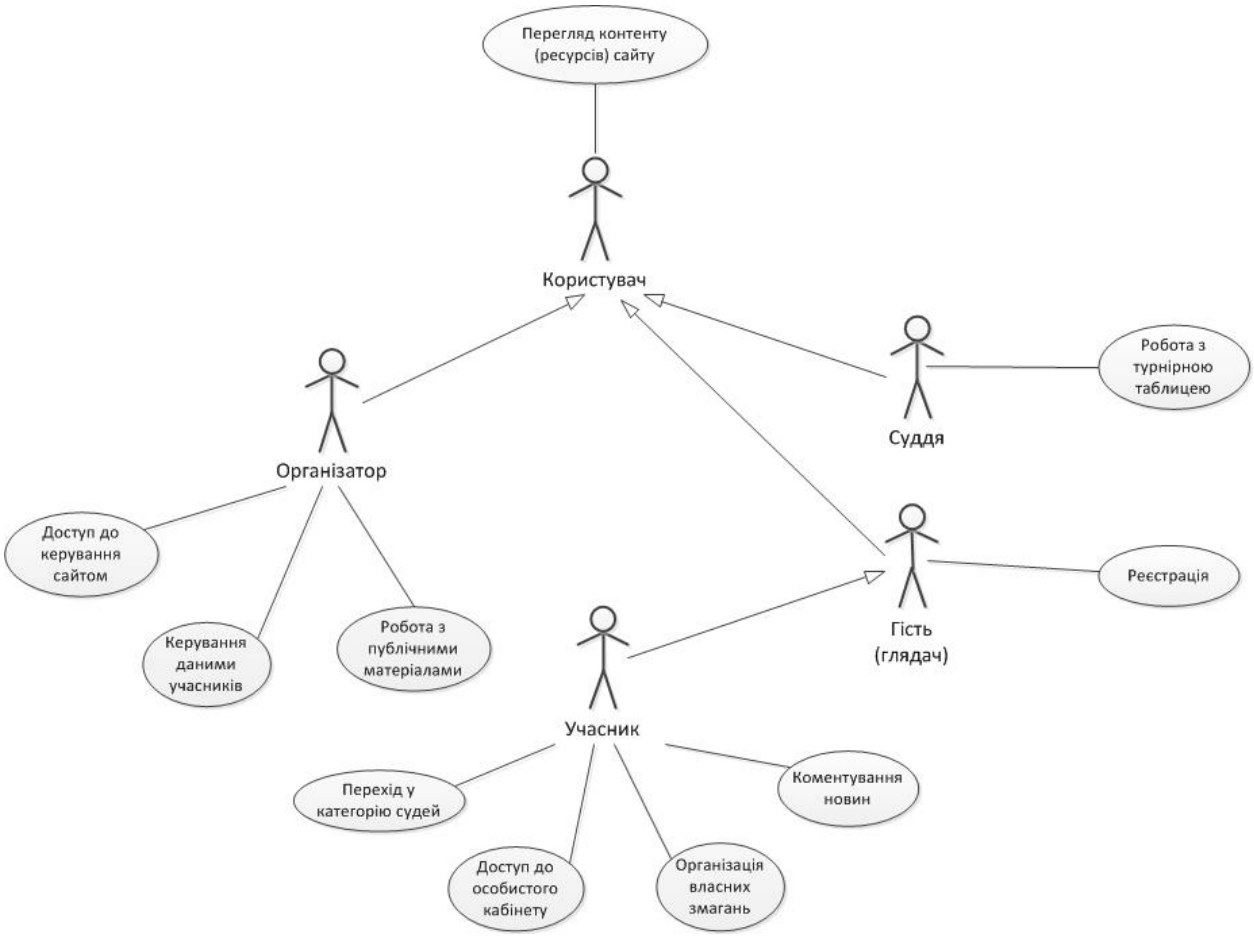


Рис. 2.1 – Загальна схема прецедентів для ролі головного адміністратора

2.2. Прецеденти для ролі гостя

Нижче описані процедури для ролі користувача з вказаними передумовами, результатом, виключними ситуаціями та детальним описом послідовності дій.

Перецент №1 Реєстрація у змаганнях

Назва: Реєстрація у змаганнях.

Учасники: гість, система.

Передумови: гість не зареєстрований у системі.

Результат: зареєстровано нового учасника змагань.

Основний сценарій:

1. Гість заповнює анкету своїми даними на сайті. Для підтвердження особи він прикріплює фотографію або скан будь-якого документу, що засвідчує особу.

2. Гість надсилає анкету організаторам.

3. Автоматично створюється неактивний профіль учасника.

4. Система надсилає на адресу електронної пошти гостя запит про підтвердження реєстрації.

5. Гість підтверджує факт реєстрації у змаганнях.

6. Система створює для учасника особистий електронний кабінет.

7. Учасник отримує персональний ідентифікаційний номер та пароль для подальшого доступу до особистого кабінету.

8. Система активує профіль учасника.

Виключні ситуації:

1. Гість невірно заповнив поля анкети або залишив їх порожніми.

2. Гість не підтвердив факт реєстрації у змаганнях.

3. Порушено зв’язок між сервером із базою даних та сайтом.

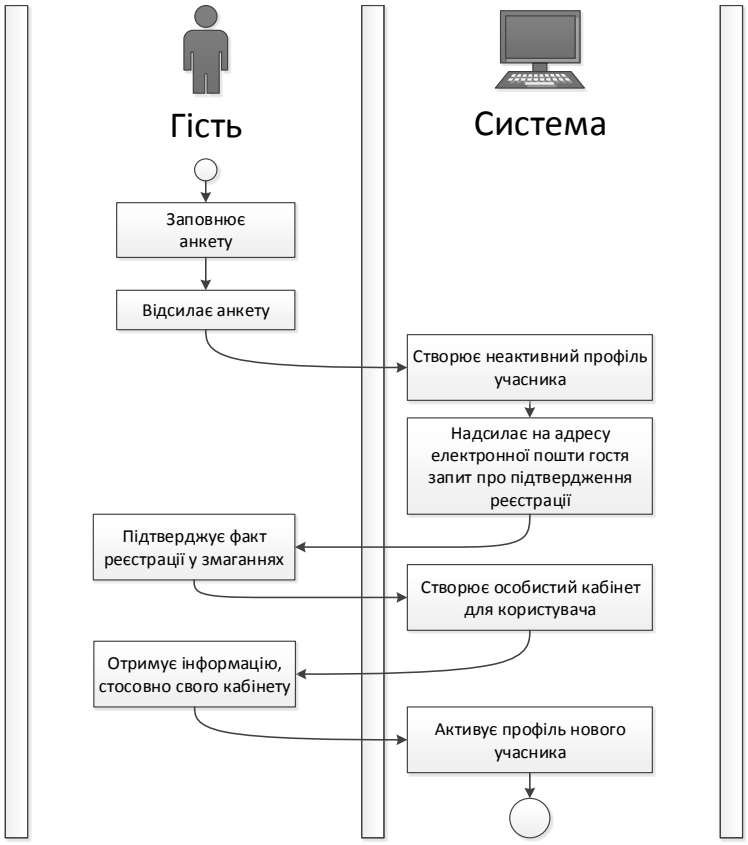


Рис. 2.2 – Схема реєстрації

Прецедент №2. Доступ до особистого кабінету

Назва: Доступ до особистого кабінету.

Учасники: учасник, система.

Передумови: учасник зареєстрований у змаганнях.

Результат: перевірено поточні дані учасника.

Сценарій:

1. Учасник входить в систему.
2. Учасник вводить ідентифікаційний номер та пароль.
3. Учасник отримує доступ до особистого кабінету.
4. Система створює сеанс роботи.
5. Учасник переглядає інформацію щодо своїх успіхів, а також місця і часу проведення наступного етапу змагань, у якому він бере участь.
6. Учасник виходить із особистого кабінету.

7. Система завершує сеанс роботи.

Виключні ситуації:

Учасник невірно ввів ідентифікаційний номер та пароль або залишив поля порожніми.

Рис. 2.3 – Схема доступу до кабінету

2.3. Діаграма бізнес-сутностей

Дана діаграма створюється на етапі бізнес моделювання. Вона відображає основні сутності та взаємозв'язки між ними. В даному випадку вона демонструє зв'язок між сутностями: організатор, суддя, гість та учасник. Діаграма бізнес-сутностей проекту зображена на рис. 2.8.

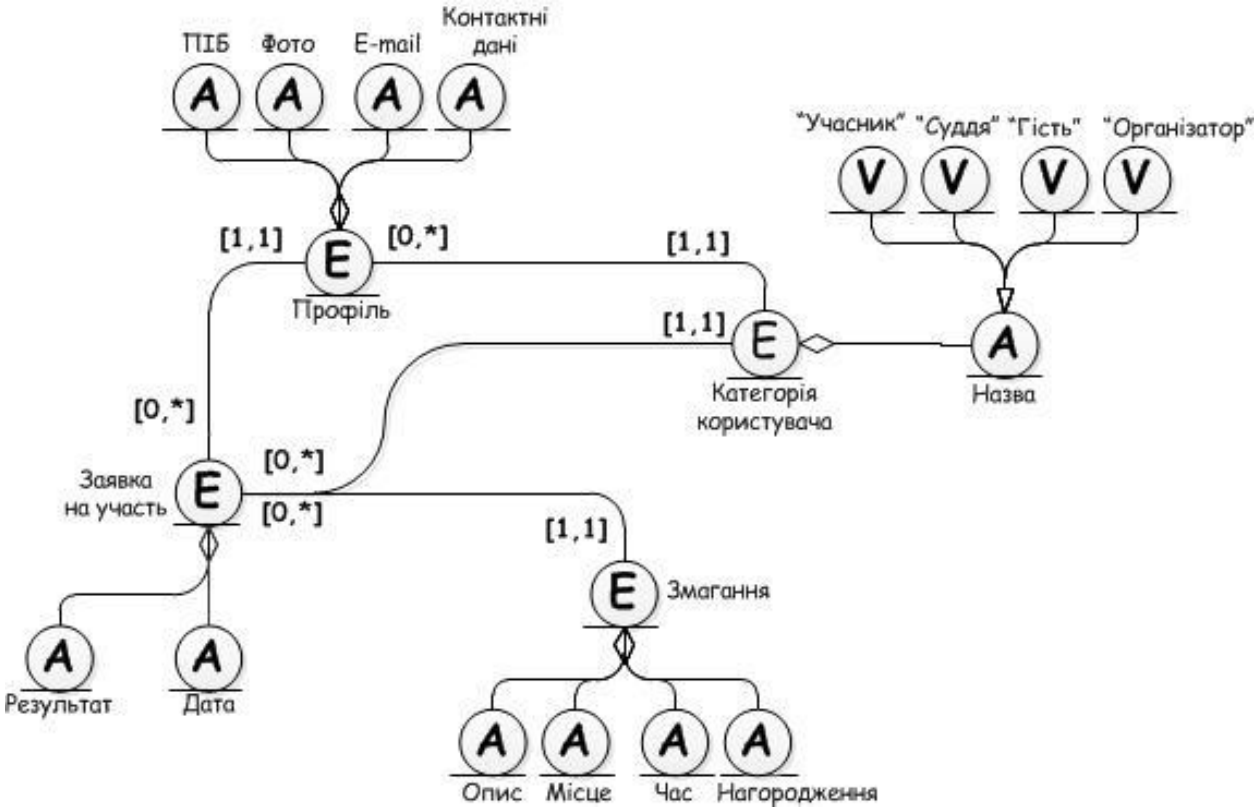


Рис. 2.8 – Діаграма бізнес-сутностей

2.4. Реляційна модель бази даних

Реляційна модель бази даних (рис 2.3) зображує структуру таблиць бази даних, взаємозв'язки між ними та поля кожної з таблиць. Наведена діаграма має багато схожого з діаграмою бізнес-сутностей. Кожній основній бізнес-сутності відповідає таблиця баз даних.

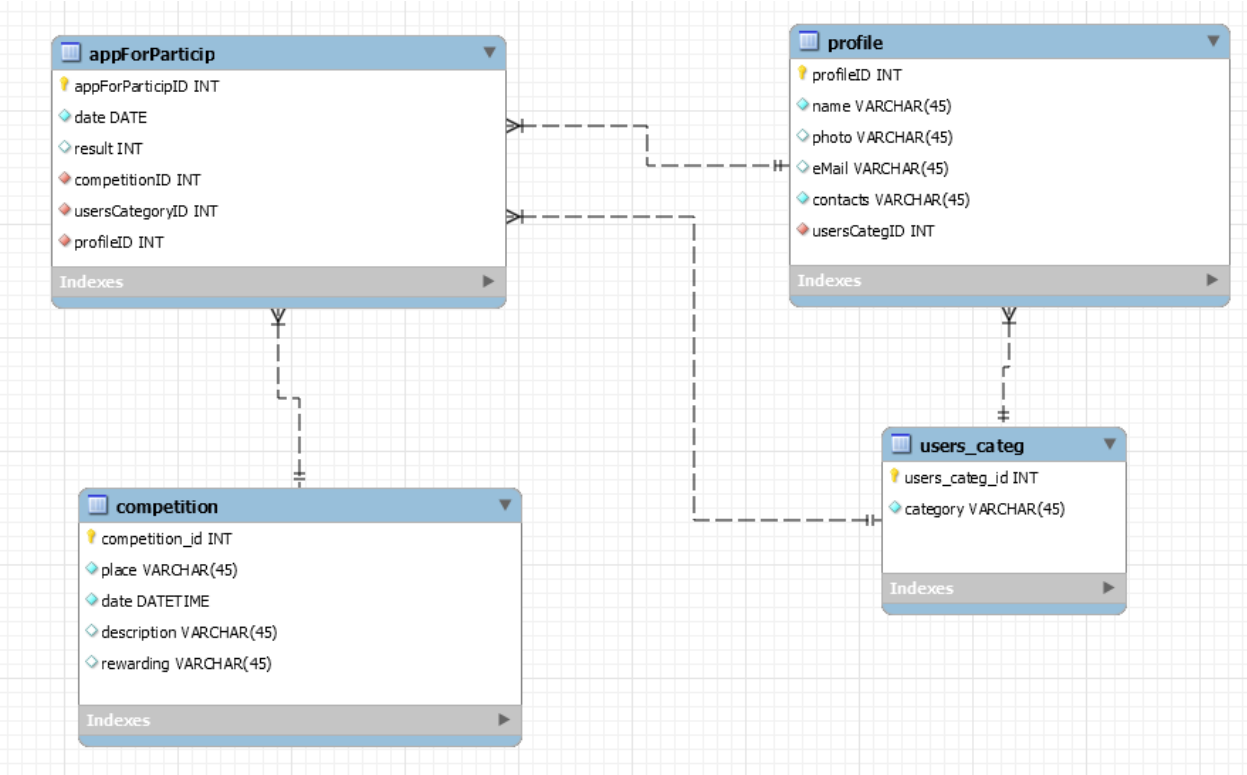


Рис 2.9 – Реляційна модель

2.5. Специфікація таблиць бази даних

Специфікація таблиць бази даних включає в себе інформацію про назви колонок таблиці, їхній тип, інформацію про те, чи є ця колонка первинним ключем, чи поле може бути пустим, чи значення поля автоматично збільшується та коментар щодо призначення колонки. Таблиці зі специфікаціями наведені нижче.

Таблиця 2.1

Таблиця **appForParticip (**application for participation**)**

appForParticip	date	result	competitionID	usersCategoryID	profileID
1	2014-11-15	NULL	1	1	1
2	2014-11-19	NULL	2	1	1
3	2014-11-15	NULL	1	2	2
4	2014-11-19	NULL	2	2	2
5	2014-11-15	NULL	1	3	3
6	2014-11-19	NULL	2	3	3
7	2014-11-15	NULL	1	4	4
8	2014-11-19	NULL	2	4	4

Таблиця 2.2

Таблиця **Profile**

profileID	name	photo	e-mail	contacts	UsersCategID
1	Pavluchkov Vlad	NULL	NULL	0981223425	1
2	Zmeul Evgeniy	NULL	NULL	0671234569	2
3	Morozov Maks	NULL	NULL	0671234565	3
4	Popenko Ruslan	NULL	NULL	0984582499	4

Таблиця 2.3

Таблиця **competition**

competition_id	Place	Date	Description	Rewarding
1	Stadium Start	2014-11-15 19:00:00	NULL	NULL
2	Metro Gidropark bus station	2014-11-19 19:00:00	NULL	NULL

Таблиця 2.4

Таблиця **users_categ**

users_categ_id	category
1	participant
2	judge
3	guest
4	organizer

РОЗДІЛ 3

РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1. **Реляційно-об’єктне відображення**

Для реляційно-об’єктного відображення в програмі використовується бібліотека Hibernate. Вона надає можливість легко встановити зв’язок з будь-якою базою даних та створити відображення між об’єктно-орієнтованою моделлю та традиційною реляційною моделлю баз даних. На рис. 3.1 зображено діаграму Entity класів. Детальна специфікація (JavaDoc) наведена нижче.

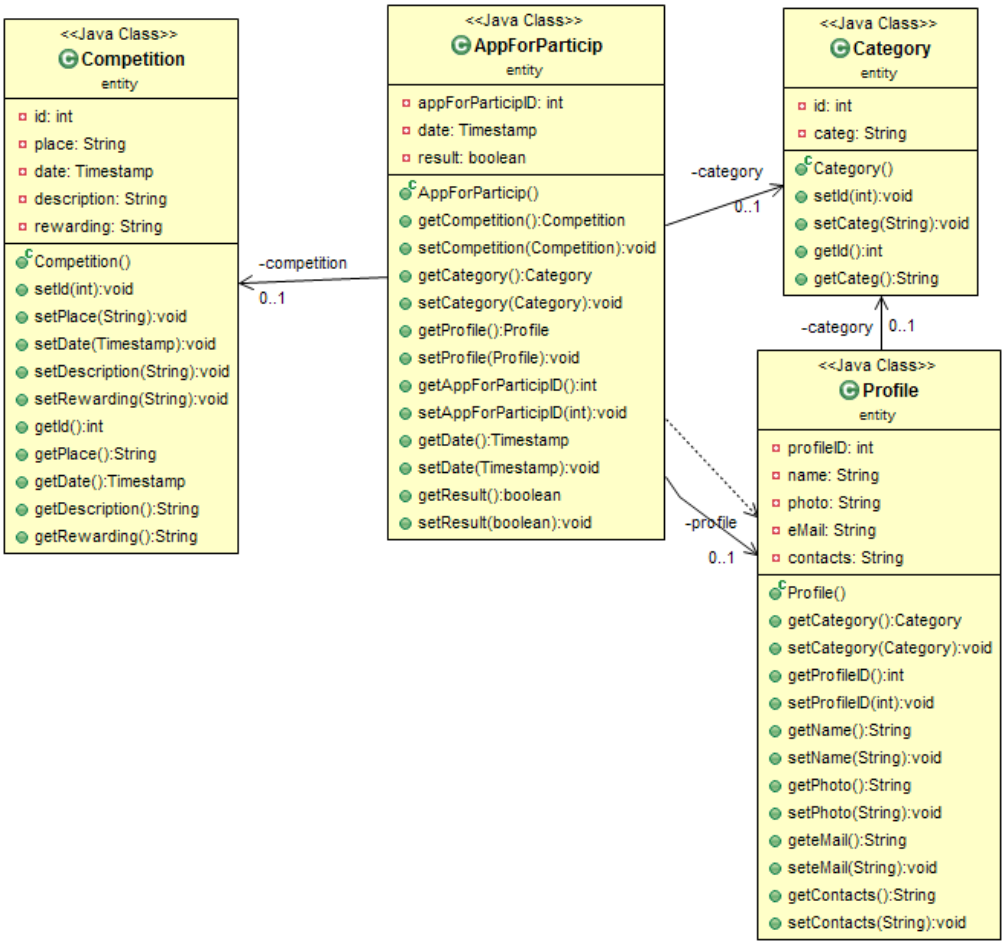


Рис 3.1 – Діаграма entity класів

3.1.1. Клас «AppForParticip»

```
@Entity
public class AppForParticip
extends java.lang.Object
```

Клас представляє сутність Заявка на участь

Since:
2015-04-14

Version:
1.0

Author:
Руслан Попенко

Field Summary

Fields

Modifier and Type	Field and Description
`private int`	`appForParticipID` Ідентифікатор заявки на участь
`private Category`	`category` Ідентифікатор категорії користувача
`private Competition`	`competition` Ідентифікатор змагання
`private java.sql.Timestamp`	`date` Дата створення заявки
`private Profile`	`profile` Ідентифікатор профілю
`private boolean`	`result` Результат реєстрації заявки

Constructor Summary

Constructors

Constructor and Description
`AppForParticip()`

Method Summary

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

Modifier and Type	Method and Description
`int`	`getAppForParticipID()`
`Category`	`getCategory()`
`Competition`	`getCompetition()`
`java.sql.Timestamp`	`getDate()`
`Profile`	`getProfile()`
`boolean`	`getResult()`
`void`	`setAppForParticipID(int appForParticipID)`
`void`	`setCategory(Category category)`
`void`	`setCompetition(Competition competition)`
`void`	`setDate(java.sql.Timestamp date)`
`void`	`setProfile(Profile profile)`
`void`	`setResult(boolean result)`

3.1.2. Клас «Category»

```
@Entity
public class Category
extends java.lang.Object
```

Клас представляє сутність Категорія

Since:

2015-04-14

Version:

1.0

Author:

Руслан Попенко

Field Summary

Fields

Modifier and Type	Field and Description
private java.lang.String	categ Категорія користувача
private int	id Ідентифікатор категорії

Constructor Summary

Constructors

Constructor and Description
Category()

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
java.lang.String	getCateg()
int	getId()
void	setCateg(java.lang.String categ)
void	setId(int id)

3.1.3. Клас «Competition»

```
@Entity
public class Competition
extends java.lang.Object
```

Клас представляє сутність Змагання

Since:

2015-04-14

Version:

1.0

Author:

Руслан Попенко

Fields

Modifier and Type	Field and Description
private java.sql.Timestamp	**date** Дата змагання
private java.lang.String	**description** Опис змагання
private int	**id** Ідентифікатор змагання
private java.lang.String	**place** Місце змагання
private java.lang.String	**rewarding** Нагородження учасників змаганяя

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
java.sql.Timestamp	**getDate**()
java.lang.String	**getDescription**()
int	**getId**()
java.lang.String	**getPlace**()
java.lang.String	**getRewarding**()
void	**setDate**(java.sql.Timestamp date)
void	**setDescription**(java.lang.String description)
void	**setId**(int id)
void	**setPlace**(java.lang.String place)
void	**setRewarding**(java.lang.String rewarding)

3.1.4. Клас «Profile»

```
@Entity
public class Profile
extends java.lang.Object
```

Клас представляє сутність Профіль

Since:

2015-04-14

Version:

1.0

Author:

Руслан Попенко

Fields

Modifier and Type	Field and Description
private Category	**category** Ідентифікатор категорії
private java.lang.String	**contacts** Контактний телефон
private java.lang.String	**eMail** Пошта
private java.lang.String	**name** Ім'я
private java.lang.String	**photo** Фото
private int	**profileID** Ідентифікатор профіля

Constructor Summary

Constructors

Constructor and Description
Profile()

3.2. Специфікація HibernateUntil класу

Клас HibernateUtil слугує для того, щоб при взаємодії із файлами конфігурації створювати hibernate-сесію. Діаграму цього інтерфейса можна побачити на рис. 3.2. Детальна специфікація (JavaDoc) наведена нижче.

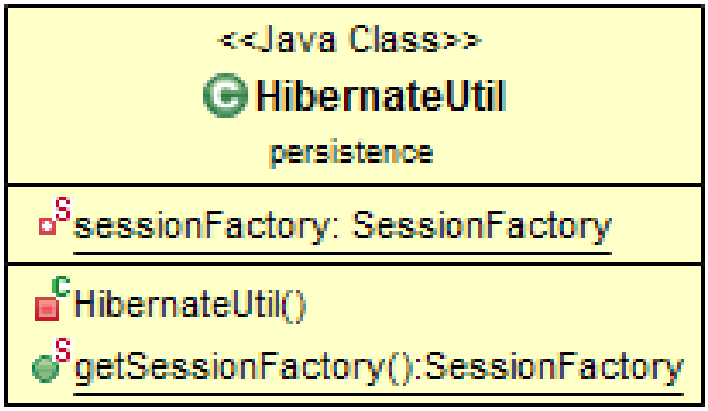


Рис 3.2. – Діаграма HibernateUnit класу

3.2.1. Клас «HibernateUntil»

public class **HibernateUtil**
extends java.lang.Object

Клас для взаємодії з конфіг файлами і створення обєкту SessionFactory,котрий відповідає за створення hibernate-сесії

Since:
2015-04-14

Version:
1.0

Author:
Руслан Попенко

Field Summary

Fields

Modifier and Type	Field and Description
private static org.hibernate.SessionFactory	**sessionFactory** об'єкт SessionFactory,котрий відповідає за створення hibernate-сесії

Constructor Summary

Constructors

Modifier	Constructor and Description
private	**HibernateUtil**()

Method Summary

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

Modifier and Type	Method and Description
static org.hibernate.SessionFactory	**getSessionFactory**() Повертає обєкт SessionFactory

3.3. Специфікація DAO-класів

Класи, що тут представлені, містять методи для роботи з базою даних. Ці методи використовують бібліотеку Hibernate. Діаграму цих класів можна побачити на рис. 3.3. Детальна специфікація наведена в додатку В.

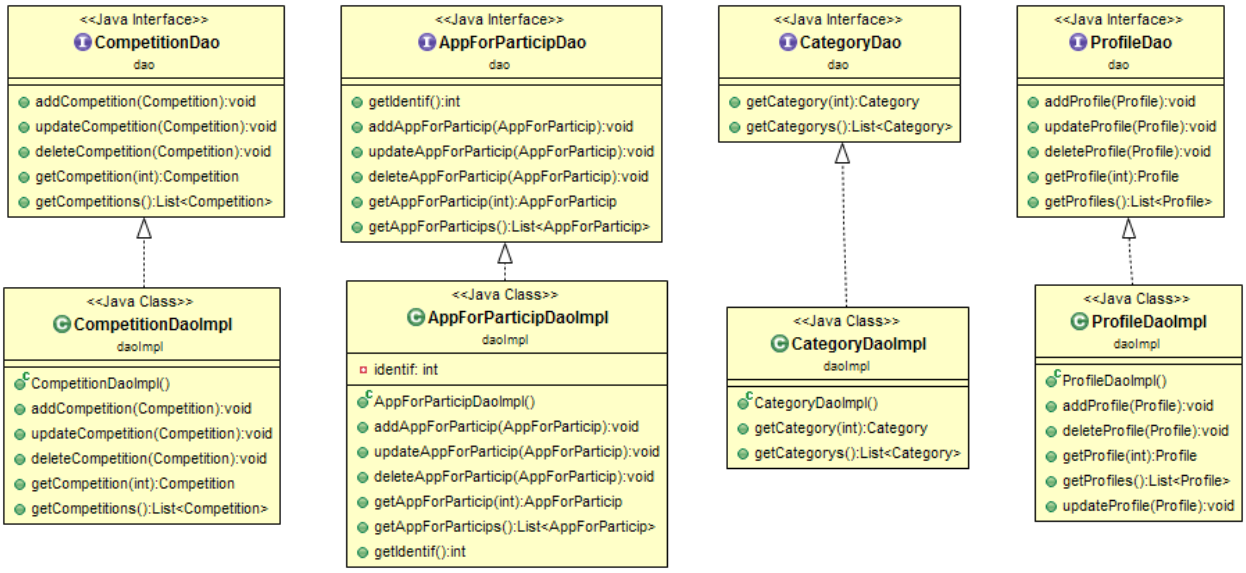


Рис 3.3 – Діаграма DAO класів

3.4. Класи контролерів та їх специфікація

Дані класи призначені для створення зв'язку між сервером та клієнтом. Діаграму цих класів можна побачити на рис. 3.4. Детальна специфікація наведена в додатку В.

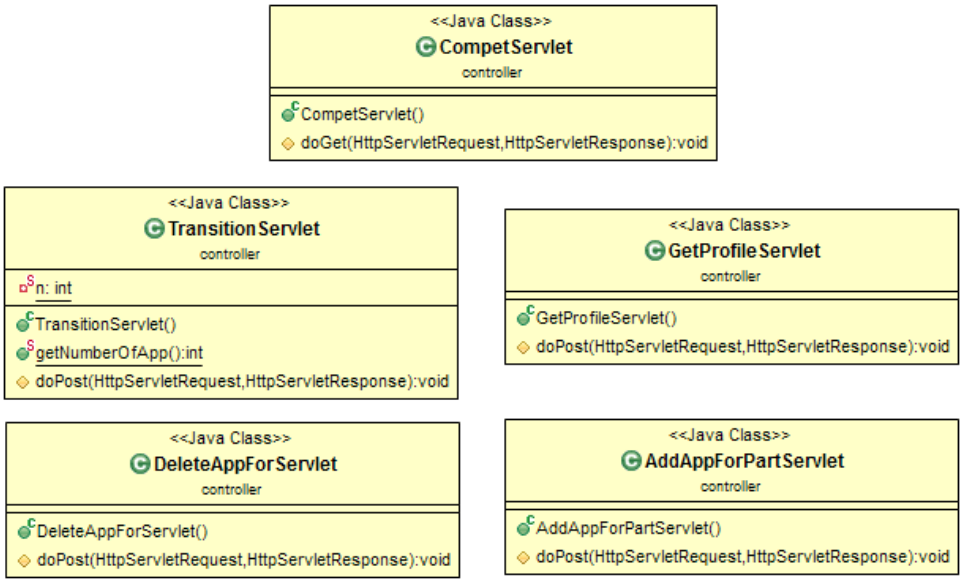


Рис. 3.4 – Діаграма класів контролерів

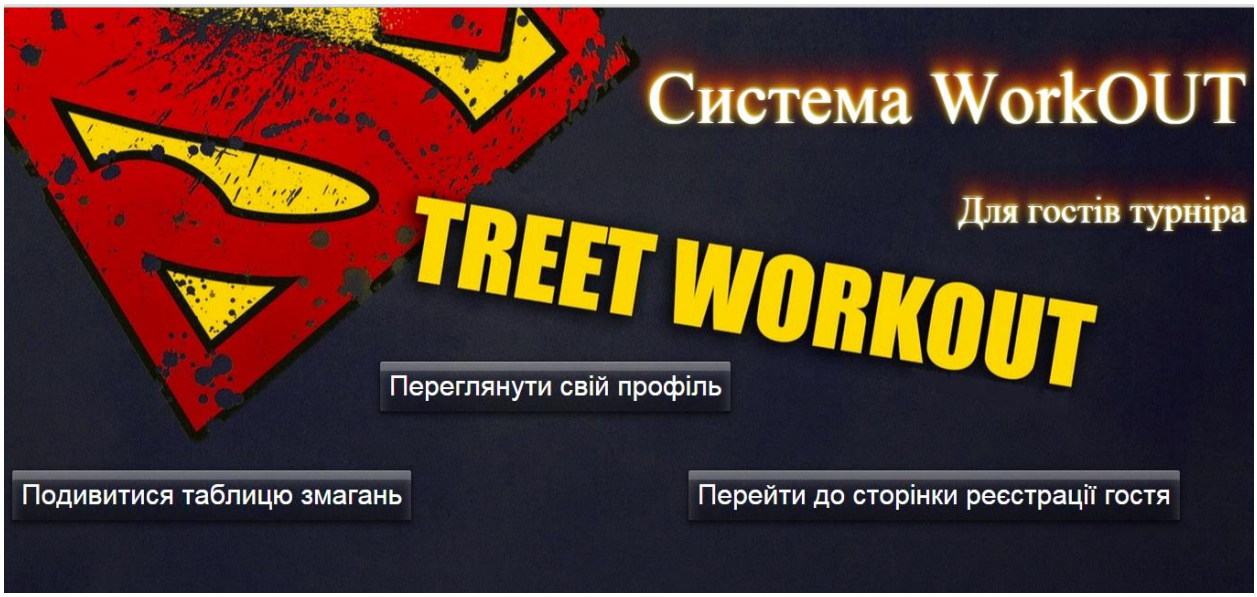
РОЗДІЛ 4

ІЛЮСТРАЦІЯ РОБОТИ ПРОГРАМИ

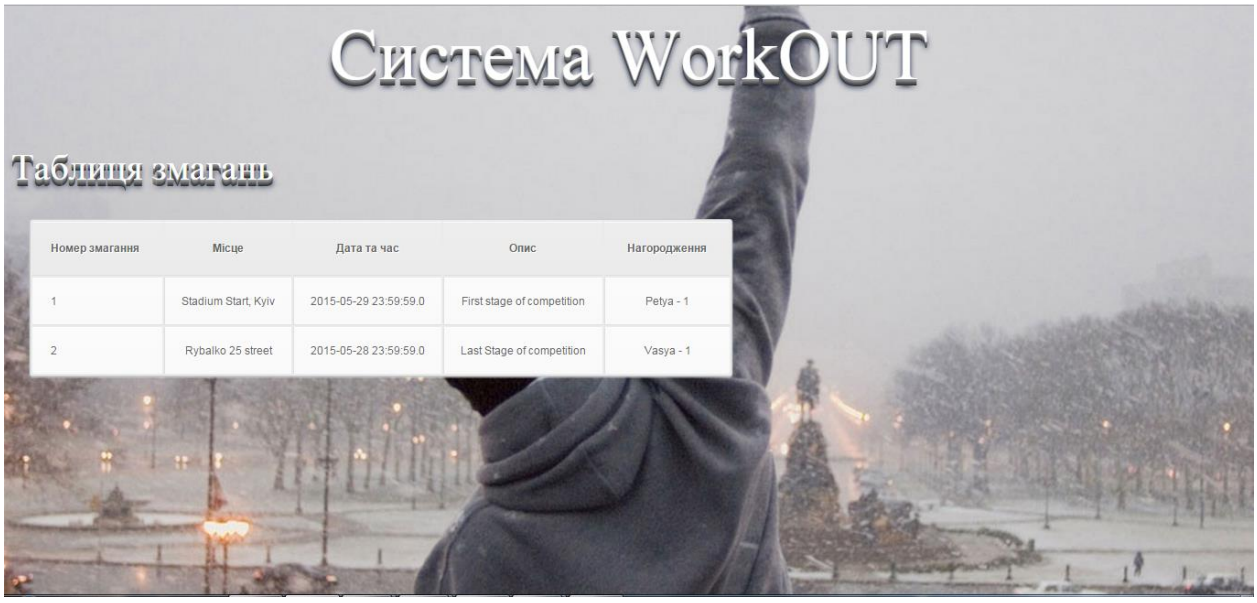
Для ілюстрації роботи програми в цьому розділі наведено графічні сценарії роботи проекту.

4.1. **Взаємодія гостя турніра і системи.**

4.1.1. Головна сторінка, на якій можна зробити вибір, яку дію потрібно зробити користувачу.

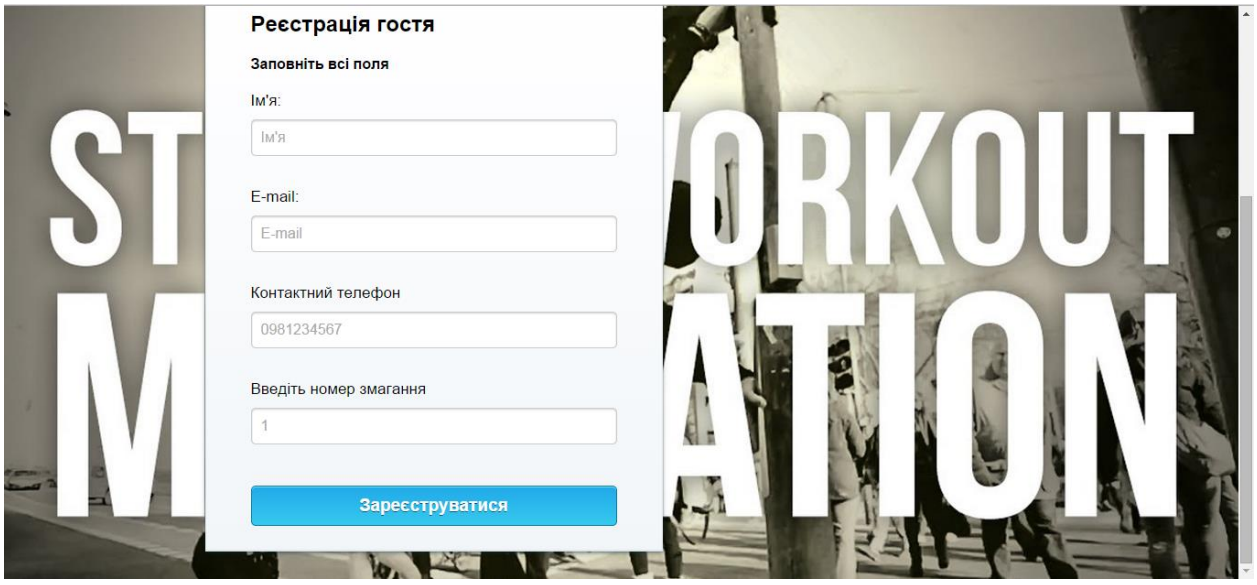


4.1.2. Перегляд турнірної таблиці.



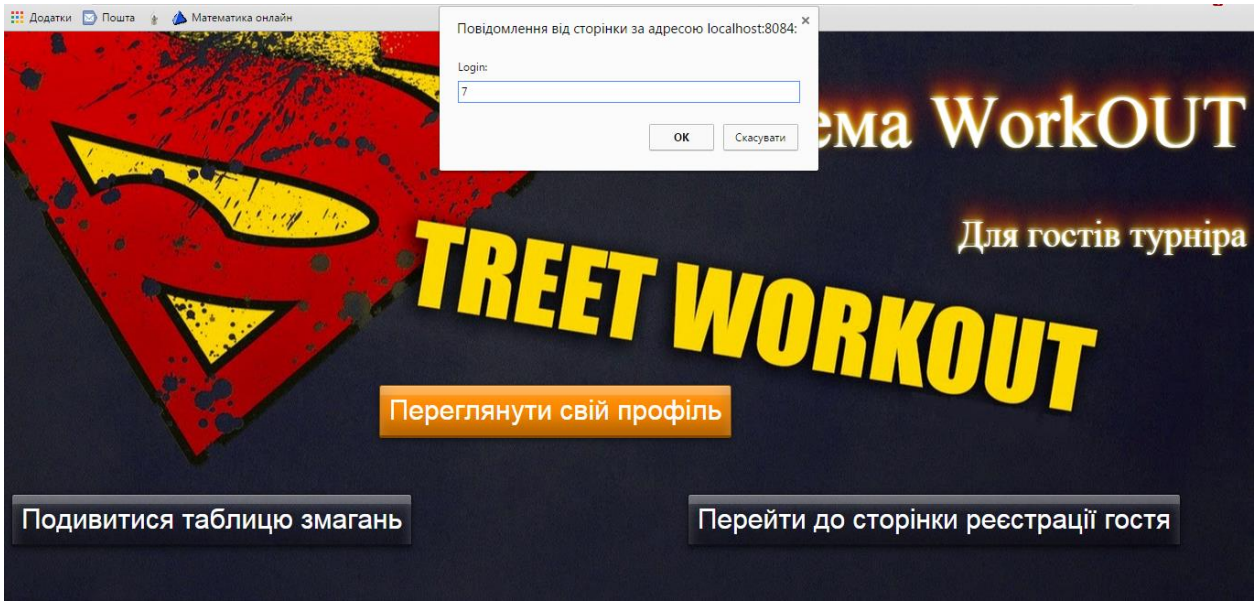
4.1.3. Реєстрація гостя. На сторінці зроблений годинник, для того щоб користувач мі г подивитися час реєстрації.

(продовження форми)

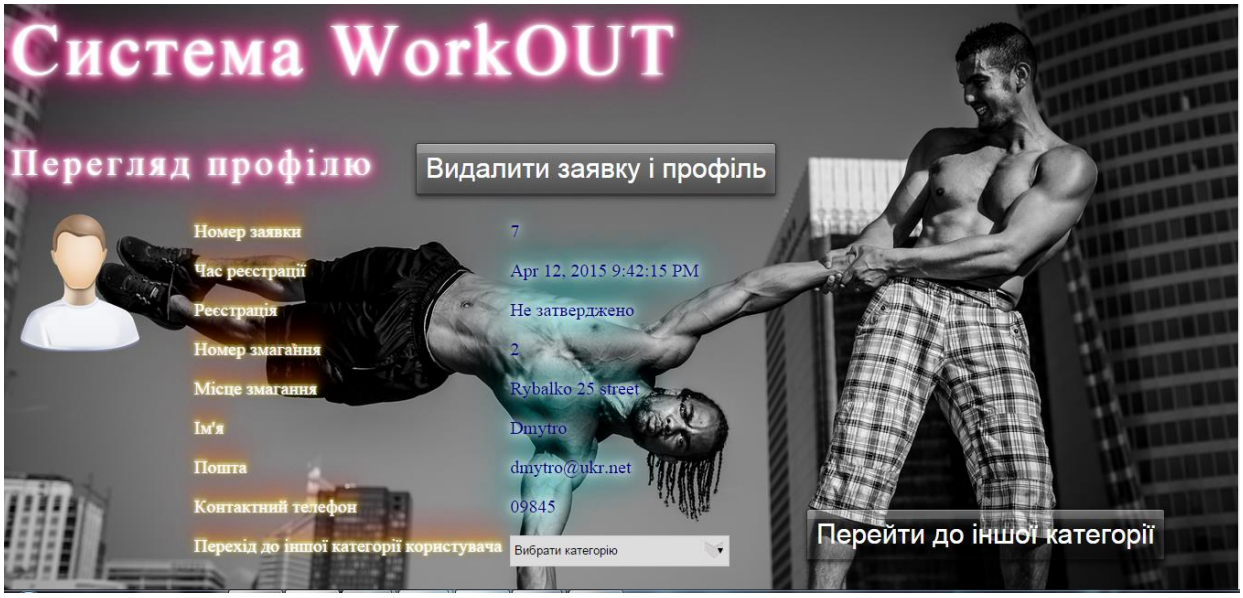


4.1.4. Перегляд заявки і профіля гостя.

4.1.4.1. Спочатку треба пройти авторизацію



4.1.4.2. Потім з’явиться інформація про гостя.

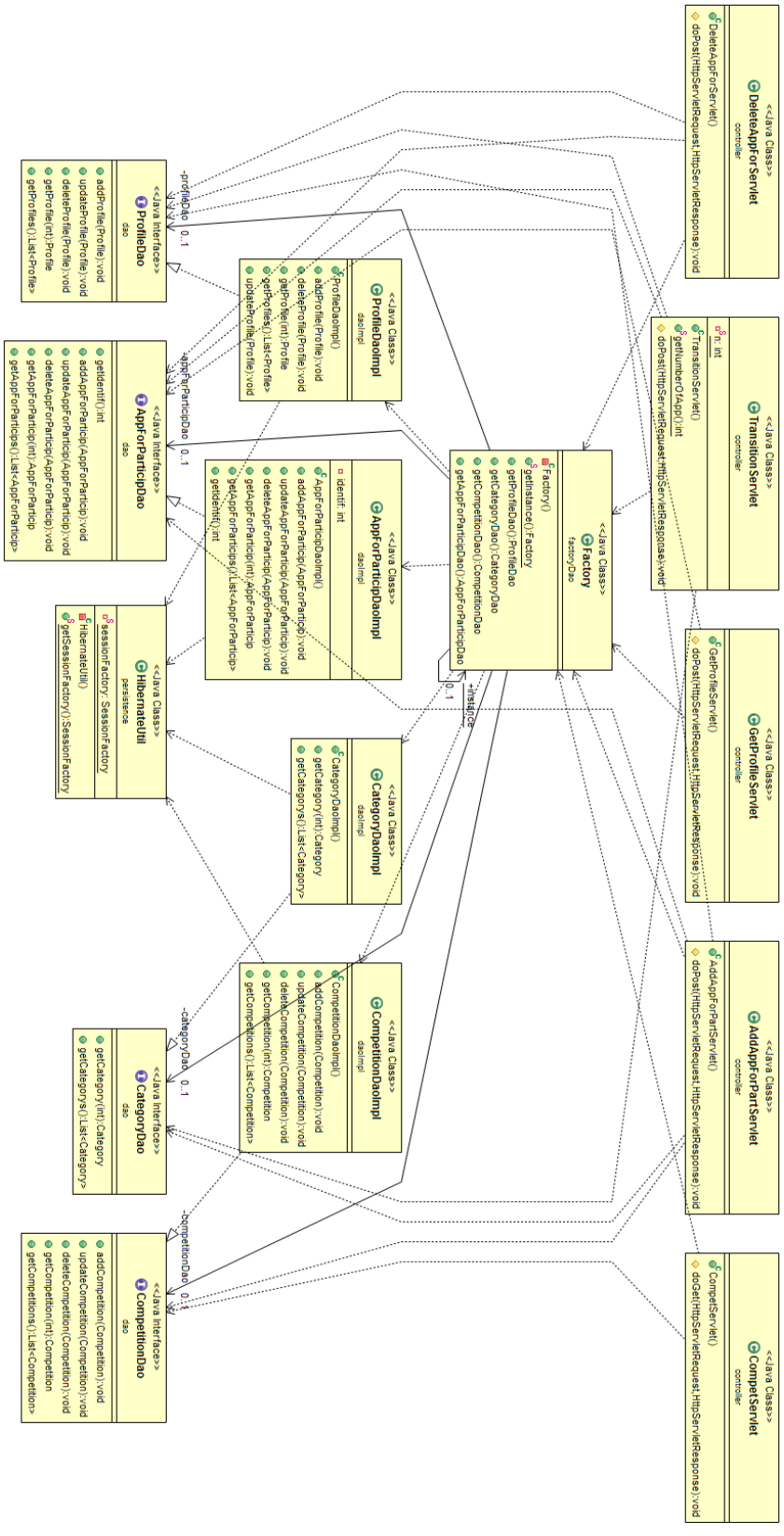


СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Apache Tomcat. – Посилання: http://tomcat.apache.org
2. Hibernate. – Посилання: http://hibernate.org
3. Maven. – Посилання: https://maven.apache.org
4. Git. – Посилання: http://git-scm.com
5. GitHub. – Посилання: https://github.com

ДОДАТОК А

Діаграма класів



ДОДАТОК Б

SQL код для створення таблиць бази даних:

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
-- -----------------------------------------------------
-- Schema compDB
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `compDB` ;
-- -----------------------------------------------------
-- Schema compDB
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `compDB` DEFAULT CHARACTER SET utf8 ;
SHOW WARNINGS;
USE `compDB` ;
-- -----------------------------------------------------
-- Table `users_categ`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `users_categ` ;

SHOW WARNINGS;
CREATE TABLE IF NOT EXISTS `users_categ` (
  `users_categ_id` INT NOT NULL,
  `category` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`users_categ_id`))
ENGINE = InnoDB;

SHOW WARNINGS;
-- -----------------------------------------------------
-- Table `profile`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `profile` ;

SHOW WARNINGS;
CREATE TABLE IF NOT EXISTS `profile` (
  `profileID` INT NOT NULL,
  `name` VARCHAR(45) NOT NULL,
  `photo` VARCHAR(45) NULL,
  `eMail` VARCHAR(45) NULL,
  `contacts` VARCHAR(45) NOT NULL,
  `usersCategID` INT NOT NULL,
  PRIMARY KEY (`profileID`),
  CONSTRAINT `fk_Profile_Users category`
    FOREIGN KEY (`usersCategID`)
    REFERENCES `users_categ` (`users_categ_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;

SHOW WARNINGS;
CREATE INDEX `fk_Profile_Users category_idx` ON `profile` (`usersCategID` ASC);

SHOW WARNINGS;
-- -----------------------------------------------------
-- Table `competition`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `competition` ;

SHOW WARNINGS;
CREATE TABLE IF NOT EXISTS `competition` (
  `competition_id` INT NOT NULL,
  `place` VARCHAR(45) NOT NULL,
  `date` DATETIME NOT NULL,
  `description` VARCHAR(45) NULL,
  `rewarding` VARCHAR(45) NULL,
  PRIMARY KEY (`competition_id`))
ENGINE = InnoDB;

SHOW WARNINGS;
-- -----------------------------------------------------
-- Table `appForParticip`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `appForParticip` ;

SHOW WARNINGS;
CREATE TABLE IF NOT EXISTS `appForParticip` (
  `appForParticipID` INT NOT NULL,
  `date` DATE NOT NULL,
  `result` INT NULL,
  `competitionID` INT NOT NULL,
  `usersCategoryID` INT NOT NULL,
  `profileID` INT NOT NULL,
  PRIMARY KEY (`appForParticipID`),
  CONSTRAINT `fk_Application for participation_Competition1`
    FOREIGN KEY (`competitionID`)
    REFERENCES `competition` (`competition_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Application for participation_Users category1`
    FOREIGN KEY (`usersCategoryID`)
    REFERENCES `users_categ` (`users_categ_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Application for participation_Profile1`
    FOREIGN KEY (`profileID`)
    REFERENCES `profile` (`profileID`)
```

```
  ON DELETE NO ACTION
  ON UPDATE NO ACTION)
ENGINE = InnoDB;

SHOW WARNINGS;
CREATE INDEX `fk_Application for participation_Competition1_idx` ON `appForParticip`
(`competitionID` ASC);

SHOW WARNINGS;
CREATE INDEX `fk_Application for participation_Users category1_idx` ON `appForParticip`
(`usersCategoryID` ASC);

SHOW WARNINGS;
CREATE INDEX `fk_Application for participation_Profile1_idx` ON `appForParticip` (`profileID` ASC);

SHOW WARNINGS;
```

Б) заповнення таблиць даними

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
-- -----------------------------------------------------
-- Data for table `users_categ`
-- -----------------------------------------------------
START TRANSACTION;
USE `compDB`;
INSERT INTO `users_categ` (`users_categ_id`, `category`) VALUES (1, 'participant');
INSERT INTO `users_categ` (`users_categ_id`, `category`) VALUES (2, 'judge');
INSERT INTO `users_categ` (`users_categ_id`, `category`) VALUES (3, 'guest');
INSERT INTO `users_categ` (`users_categ_id`, `category`) VALUES (4, 'organizer');

COMMIT;
-- -----------------------------------------------------
-- Data for table `profile`
-- -----------------------------------------------------
START TRANSACTION;
USE `compDB`;
INSERT INTO `profile` (`profileID`, `name`, `photo`, `eMail`, `contacts`, `usersCategID`) VALUES (1,
'Pavluchkov Vladislav', NULL, NULL, '0631234567', 4);
INSERT INTO `profile` (`profileID`, `name`, `photo`, `eMail`, `contacts`, `usersCategID`) VALUES (2,
'Zmeul Evgeniy', NULL, NULL, '0671234567', 2);
INSERT INTO `profile` (`profileID`, `name`, `photo`, `eMail`, `contacts`, `usersCategID`) VALUES (3,
'Morozov Max', NULL, NULL, '0961234567', 3);
INSERT INTO `profile` (`profileID`, `name`, `photo`, `eMail`, `contacts`, `usersCategID`) VALUES (4,
'Popenko Ruslan', NULL, NULL, '0981234567', 1);
INSERT INTO `profile` (`profileID`, `name`, `photo`, `eMail`, `contacts`, `usersCategID`) VALUES (5,
'Korchak Myhailo', NULL, NULL, '0501234567', 3);
```

```
COMMIT;
-- -----------------------------------------------------
-- Data for table `competition`
-- -----------------------------------------------------
START TRANSACTION;
USE `compDB`;
INSERT INTO `competition` (`competition_id`, `place`, `date`, `description`, `rewarding`) VALUES (1,
'Stadium Start', '2014-11-15 19:00:00', NULL, NULL);
INSERT INTO `competition` (`competition_id`, `place`, `date`, `description`, `rewarding`) VALUES (2,
'Metro Gidropark bus station', '2014-11-19 19:00:00', NULL, NULL);

COMMIT;
-- -----------------------------------------------------
-- Data for table `appForParticip`
-- -----------------------------------------------------
START TRANSACTION;
USE `compDB`;
INSERT INTO `appForParticip` (`appForParticipID`, `date`, `result`, `competitionID`,
`usersCategoryID`, `profileID`) VALUES (1, '2014-11-15', NULL, 1, 1, 1);
INSERT INTO `appForParticip` (`appForParticipID`, `date`, `result`, `competitionID`,
`usersCategoryID`, `profileID`) VALUES (2, '2014-11-19', NULL, 2, 1, 1);
INSERT INTO `appForParticip` (`appForParticipID`, `date`, `result`, `competitionID`,
`usersCategoryID`, `profileID`) VALUES (3, '2014-11-15', NULL, 1, 2, 2);
INSERT INTO `appForParticip` (`appForParticipID`, `date`, `result`, `competitionID`,
`usersCategoryID`, `profileID`) VALUES (4, '2014-11-19', NULL, 2, 2, 2);
INSERT INTO `appForParticip` (`appForParticipID`, `date`, `result`, `competitionID`,
`usersCategoryID`, `profileID`) VALUES (5, '2014-11-15', NULL, 1, 3, 3);
INSERT INTO `appForParticip` (`appForParticipID`, `date`, `result`, `competitionID`,
`usersCategoryID`, `profileID`) VALUES (6, '2014-11-19', NULL, 2, 3, 3);
INSERT INTO `appForParticip` (`appForParticipID`, `date`, `result`, `competitionID`,
`usersCategoryID`, `profileID`) VALUES (7, '2014-11-15', NULL, 1, 4, 4);
INSERT INTO `appForParticip` (`appForParticipID`, `date`, `result`, `competitionID`,
`usersCategoryID`, `profileID`) VALUES (8, '2014-11-19', NULL, 2, 4, 4);

COMMIT;
-- -----------------------------------------------------
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

ДОДАТОК В

HibernateUtil.java

```
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
/**
 * Клас для взаємодії з конфіг файлами і створення обєкту
SessionFactory,котрий відповідає за створення hibernate-сесії
 * @author Руслан Попенко
 * @version 1.0
 * @since 2015-04-14
 */
public class HibernateUtil {
   /**
    * об'єкт SessionFactory,котрий відповідає за створення
hibernate-сесії
    */
   private static SessionFactory sessionFactory;


   private HibernateUtil () {

   }

   /**
    * Створює нову сесію із hibernate.cfg.xml
    */
   static {
      try {
         sessionFactory = new
Configuration().configure().buildSessionFactory();
      } catch (Throwable e){
         throw new ExceptionInInitializerError(e);
      }

   }

   /**
    * Повертає обєкт SessionFactory
    * @return обєкт SessionFactory
    */
   public static SessionFactory getSessionFactory() {
      return sessionFactory;
   }
}
```

AppForParticipDaoImpl.java

```
package daoImpl;
import java.sql.SQLException;
```

```
import java.util.List;

import org.hibernate.Session;

import entity.AppForParticip;

import persistence.HibernateUtil;
import dao.AppForParticipDao;
public class AppForParticipDaoImpl implements AppForParticipDao{
   private int identif;

   @Override
   public void addAppForParticip(AppForParticip appForParticip)
throws SQLException {
      Session session = null;
      try {
          session =
HibernateUtil.getSessionFactory().openSession();
          session.beginTransaction();
          session.save(appForParticip);
          identif=appForParticip.getAppForParticipID();
          session.getTransaction().commit();

      } catch (Exception e){
         e.printStackTrace();
      } finally {
         if ((session!= null) && (session.isOpen()))
            session.close();
      }

   }

   @Override
   public void updateAppForParticip(AppForParticip
appForParticip) throws SQLException {
      Session session = null;
      try {
          session =
HibernateUtil.getSessionFactory().openSession();
          session.beginTransaction();
          session.update(appForParticip);
          session.getTransaction().commit();

      } catch (Exception e){
         e.printStackTrace();
      } finally {
```

```
         if ((session!= null) && (session.isOpen()))
            session.close();
      }

   }

   @Override
   public void deleteAppForParticip(AppForParticip
appForParticip) throws SQLException {
      Session session = null;
      try {
          session =
HibernateUtil.getSessionFactory().openSession();
          session.beginTransaction();
          session.delete(appForParticip);
          session.getTransaction().commit();

      } catch (Exception e){
         e.printStackTrace();
      } finally {
         if ((session!= null) && (session.isOpen()))
            session.close();
      }

   }

   @Override
   public AppForParticip getAppForParticip(int id) throws
SQLException {
      AppForParticip result=null;

      Session session = null;
      try {
          session =
HibernateUtil.getSessionFactory().openSession();
          result = (AppForParticip)
session.get(AppForParticip.class, id);

      } catch (Exception e){
         e.printStackTrace();
      } finally {
         if ((session!= null) && (session.isOpen()))
            session.close();
      }
      return result;
   }
```

```
   @Override
   public List<AppForParticip> getAppForParticips() throws
SQLException {
      List<AppForParticip> appForParticips = null;
      Session session = null;
      try {
          session =
HibernateUtil.getSessionFactory().openSession();
          appForParticips =
session.createCriteria(AppForParticip.class).list();

      } catch (Exception e){
         e.printStackTrace();
      } finally {
         if ((session!= null) && (session.isOpen()))
            session.close();
      }
      return appForParticips;
   }
   @Override
   public int getIdentif () {
      return identif;
   }

}
```

AddAppForPartServlet.java

```
package controller;

import dao.AppForParticipDao;
import dao.CategoryDao;
import dao.CompetitionDao;
import dao.ProfileDao;
import entity.AppForParticip;
import entity.Profile;
import factoryDao.Factory;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
```

```
import java.io.PrintWriter;
import java.sql.SQLException;
import java.sql.Timestamp;

/**
 * Сервлет для взаэмодії зі сторінками /registr.jsp,
/registred.jsp
 * @author Руслан Попенко
 * @version 1.0
 * @since 2015-04-14
 */
@WebServlet("/AppForPartServlet")
public class AddAppForPartServlet extends HttpServlet {

    /**
     * Отримаємо дані про профіль з /registr.jsp,
     * заносимо профіль і заявку в базу даних,
     * і пересилаємо результат реєстрації на /registred.jsp
     * @param request запит
     * @param response відповідь
     * @throws ServletException необхідне виключення
     * @throws IOException необхідне виключення
     */
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
        Factory factory=Factory.getInstance();
        ProfileDao profileDao = factory.getProfileDao();
        CategoryDao categoryDao=factory.getCategoryDao();
        CompetitionDao
competitionDao=factory.getCompetitionDao();
        AppForParticipDao
appForParticipDao=factory.getAppForParticipDao();

        String name=request.getParameter("name");
        String photo="photo";
        String eMail=request.getParameter("eMail");
        String contact=request.getParameter("contact");
        Profile profile=new Profile();
        try {
            profile.setCategory(categoryDao.getCategory(3));
        } catch (SQLException e) {
            e.printStackTrace();
        }
        profile.setName(name);
        profile.setPhoto(photo);
```

```
        profile.seteMail(eMail);
        profile.setContacts(contact);

        try {
            profileDao.addProfile(profile);
        } catch (SQLException e) {
            e.printStackTrace();
        }


        String number=request.getParameter("num");
        int num=Integer.parseInt(number);



        AppForParticip appForParticip=new AppForParticip();

        try {

appForParticip.setCategory(categoryDao.getCategory(3));
        } catch (SQLException e) {
            e.printStackTrace();
        }

        appForParticip.setProfile(profile);
        try {

appForParticip.setCompetition(competitionDao.getCompetition(num)
);
        } catch (SQLException e) {
            e.printStackTrace();
        }


        java.util.Date date= new java.util.Date();
        Timestamp current=new Timestamp(date.getTime());
        appForParticip.setDate(current);
        appForParticip.setResult(false);

        try {
            appForParticipDao.addAppForParticip(appForParticip);
        } catch (SQLException e) {
            e.printStackTrace();
        }

        request.setAttribute("id",
```

```
""+appForParticipDao.getIdentif());

request.getRequestDispatcher("/registred.jsp").forward(request,
response);


    }


}
```

CompetServlet.java

```
package controller;

import dao.CompetitionDao;
import entity.Competition;
import factoryDao.Factory;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import java.sql.SQLException;
import java.util.List;

/**
 * Сервлет для взаэмодії зі сторінкою /competition.jsp
 * @author Руслан Попенко
 * @version 1.0
 * @since 2015-04-14
 */
@WebServlet("/CompetServlet")
public class CompetServlet extends HttpServlet {

    /**
     * Повертає з бази даних таблицю Змагання
     * пересилає її на /competition.jsp
     * @param request запит
     * @param response відповідь
     * @throws ServletException необхідне виключення
     * @throws IOException необхідне виключення
```

```
     */
    protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
        Factory factory = Factory.getInstance();
        CompetitionDao competitionDao =
factory.getCompetitionDao();

        List<Competition> compList = null;
        try {
            compList = competitionDao.getCompetitions();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        request.setAttribute("ListOfComp", compList);

request.getRequestDispatcher("/competition.jsp").forward(request
, response);
    }

}
```

DeleteAppForServlet.java

```
package controller;

import dao.AppForParticipDao;
import dao.ProfileDao;
import entity.AppForParticip;
import entity.Profile;
import factoryDao.Factory;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;

/**
 * Сервлет для взаэмодії зі сторінками /index.jsp, /profile.jsp
```

```
 * @author Руслан Попенко
 * @version 1.0
 * @since 2015-04-14
 */
@WebServlet("/DeleteAppForServlet")
public class DeleteAppForServlet extends HttpServlet {

    /**
     * Видаляє профіль за бази даних, отриманий з /profile.jsp
     * і напрявляє на /index.jsp
     * @param request запит
     * @param response відповідь
     * @throws ServletException необхідне виключення
     * @throws IOException необхідне виключення
     */
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
        Factory factory=Factory.getInstance();
        AppForParticipDao
appForParticipDao=factory.getAppForParticipDao();
        ProfileDao profileDao = factory.getProfileDao();
        String str=request.getParameter("deleted").toString();
        AppForParticip appForParticip=new AppForParticip();

        try {

appForParticip=appForParticipDao.getAppForParticip(Integer.parse
Int(str));
        } catch (SQLException e) {
            e.printStackTrace();
        }
        Profile profile=appForParticip.getProfile();
        try {

appForParticipDao.deleteAppForParticip(appForParticip);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        try {
            profileDao.deleteProfile(profile);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
```

```
            out.println("<script type=\"text/javascript\">");
            out.println("alert('Deleted');");
            out.println("location='index.jsp';");
            out.println("</script>");

        }

}
```

GetProfileServlet.java

```
package controller;

import dao.AppForParticipDao;
import dao.CategoryDao;
import dao.CompetitionDao;
import dao.ProfileDao;
import entity.AppForParticip;
import entity.Profile;
import factoryDao.Factory;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.sql.SQLException;
import java.sql.Timestamp;

/**
 * Сервлет для взаэмодії зі сторінкою /profile.jsp
 * @author Руслан Попенко
 * @version 1.0
 * @since 2015-04-14
 */
@WebServlet("/GetProfileServlet")
public class GetProfileServlet extends HttpServlet {
    /**
     * Виймає профіль з бази даних і відправляє на /profile.jsp
     * @param request запит
     * @param response відповідь
     * @throws ServletException необхідне виключення
```

```
     * @throws IOException необхідне виключення
     */
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
        String usN =
request.getParameter("userName").toString();
        request.setAttribute("usN", usN);
        Factory factory=Factory.getInstance();

        AppForParticipDao
appForParticipDao=factory.getAppForParticipDao();

        AppForParticip ap=new AppForParticip();

        int n=Integer.parseInt(usN);
        try {
            ap=appForParticipDao.getAppForParticip(n);
        } catch (SQLException e) {

request.getRequestDispatcher("/error.jsp").forward(request,
response);
        }
        if (ap==null){

request.getRequestDispatcher("/error.jsp").forward(request,
response);
        }
        Timestamp date=ap.getDate();

        request.setAttribute("TimeOfApp",
date.toLocaleString());

        if(ap.getResult()) {
            request.setAttribute("result",
"\u0417\u0430\u0442\u0432\u0435\u0440\u0434\u0436\u0435\u043d\u0
43e");
        } else {
            request.setAttribute("result", "\u041d\u0435
\u0437\u0430\u0442\u0432\u0435\u0440\u0434\u0436\u0435\u043d\u04
3e");
        }

        String compId=ap.getCompetition().getId()+"";
        String
compDate=ap.getCompetition().getDate().toLocaleString();
```

```
        String compPlace=ap.getCompetition().getPlace();

        request.setAttribute("compId", compId);
        request.setAttribute("compDate", compDate);
        request.setAttribute("compPlace", compPlace);

        Profile profile=ap.getProfile();
        String name=profile.getName();
        String photo=profile.getPhoto();
        String eMail=profile.geteMail();
        String contacts=profile.getContacts();

        request.setAttribute("name", name);
        request.setAttribute("photo", photo);
        request.setAttribute("eMail", eMail);
        request.setAttribute("contacts", contacts);

        String category=ap.getCategory().getCateg();
        if (category.equals("guest")){
            request.setAttribute("category", category);

request.getRequestDispatcher("/profile.jsp").forward(request,
response);

        } else {

request.getRequestDispatcher("/error.jsp").forward(request,
response);
        }


    }
}
```

TransitionServlet.java

```
package controller;

import dao.AppForParticipDao;
import dao.CategoryDao;
import dao.ProfileDao;
import entity.AppForParticip;
import entity.Category;
```

```
import entity.Profile;
import factoryDao.Factory;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;

/**
 * Сервлет для взаэмодії зі сторінкою /profile.jsp
 * @author Руслан Попенко
 * @version 1.0
 * @since 2015-04-14
 */
@WebServlet("/TransitionServlet")
public class TransitionServlet extends HttpServlet {
    /**
     * Поле ідентиіфікатор заявки
     */
    private static int n;

    /**
     * Геттер для ідентиіфікатора заявки
     * @return ідентиіфікатор заявки
     */
    public static int getNumberOfApp (){
        return n;
    }

    /**
     * Здійснює перехід з гостя до іншої категорії
     * @param request запит
     * @param response відповідь
     * @throws ServletException необхідне виключення
     * @throws IOException необхідне виключення
     */
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
        Factory factory=Factory.getInstance();
        AppForParticipDao
```

```
appForParticipDao=factory.getAppForParticipDao();
        ProfileDao profileDao = factory.getProfileDao();
        CategoryDao categoryDao=factory.getCategoryDao();
        String
str=request.getParameter("numberOfApp").toString();
        n=Integer.parseInt(str);
        AppForParticip appForParticip=new AppForParticip();
        Category category=new Category();

        try {

appForParticip=appForParticipDao.getAppForParticip(Integer.parse
Int(str));
        } catch (SQLException e) {
            e.printStackTrace();
        }
        Profile profile=appForParticip.getProfile();

        String selectedValue=request.getParameter("catOptions");
        if(selectedValue.equals("1"))
        {
            try {
                category=categoryDao.getCategory(1);
            } catch (SQLException e) {
                e.printStackTrace();
            }
            appForParticip.setCategory(category);
            profile.setCategory(category);
            try {

appForParticipDao.updateAppForParticip(appForParticip);
                profileDao.updateProfile(profile);
            } catch (SQLException e) {
                e.printStackTrace();
            }

        }
        else if(selectedValue.equals("2"))
        {
            try {
                category=categoryDao.getCategory(2);
            } catch (SQLException e) {
                e.printStackTrace();
            }
            appForParticip.setCategory(category);
            profile.setCategory(category);
```

```
            try {

appForParticipDao.updateAppForParticip(appForParticip);
                profileDao.updateProfile(profile);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        else if(selectedValue.equals("4"))
        {
            try {
                category=categoryDao.getCategory(4);
            } catch (SQLException e) {
                e.printStackTrace();
            }
            appForParticip.setCategory(category);
            profile.setCategory(category);
            try {

appForParticipDao.updateAppForParticip(appForParticip);
                profileDao.updateProfile(profile);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }


        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
        out.println("<script type=\"text/javascript\">");
        out.println("alert('Category changed');");
        out.println("location='index.jsp';");
        out.println("</script>");

    }

}
```

Factory.java

```
package factoryDao;

import dao.AppForParticipDao;
import dao.CategoryDao;
import dao.CompetitionDao;
import dao.ProfileDao;
import daoImpl.AppForParticipDaoImpl;
```

```
import daoImpl.CategoryDaoImpl;
import daoImpl.CompetitionDaoImpl;
import daoImpl.ProfileDaoImpl;


public class Factory {
   public static Factory instance = new Factory();
   private ProfileDao profileDao;
   private CategoryDao categoryDao;
   private CompetitionDao competitionDao;
   private AppForParticipDao appForParticipDao;

   private Factory (){

   }

   public static Factory getInstance(){
      return Factory.instance;
   }

   public ProfileDao getProfileDao (){
      if (profileDao == null) {
         profileDao = new ProfileDaoImpl();
      }
      return profileDao;
   }

   public CategoryDao getCategoryDao (){
      if (categoryDao == null) {
         categoryDao = new CategoryDaoImpl();
      }
      return categoryDao;
   }

   public CompetitionDao getCompetitionDao (){
      if (competitionDao == null) {
         competitionDao = new CompetitionDaoImpl();
      }
      return competitionDao;
   }
   public AppForParticipDao getAppForParticipDao (){
      if (appForParticipDao == null) {
         appForParticipDao = new AppForParticipDaoImpl();
      }
      return appForParticipDao;   }
```