РОЗРОБКА ІГРОВОЇ ПРОГРАМИ

З ВИКОРИСТАННЯМ 3D ГРАФІКИ

МОВОЮ JAVA

Курсова робота

студентів групи ПМі-31

Звізло Ю. З.

Слободянюк Н. А.

Науковий керівник:

доц. Клакович Л.М.

Зміст

[Вступ 4](#_Toc483870260)

[2. Етап підготовки 5](#_Toc483870261)

[2.1. Постановка задачі 5](#_Toc483870262)

[2.2. Прийоми при проектуванні архітектури ігор 6](#_Toc483870263)

[2.2.1. Компонентно-орієнтоване програмування (КОП) 6](#_Toc483870264)

[2.2.2. Машина станів та дерево поведінки 8](#_Toc483870265)

[2.3. Шаблони програмування 8](#_Toc483870266)

[2.3.1. MVC 9](#_Toc483870267)

[2.3.2. Репозиторій 10](#_Toc483870268)

[2.3.3. Стратегія 11](#_Toc483870269)

[2.3.4. Шаблонний метод 12](#_Toc483870270)

[3. Огляд використаних технологій 14](#_Toc483870271)

[3.1. Засоби для бази даних 14](#_Toc483870272)

[3.2. Засоби для інфраструктури та контролерів 14](#_Toc483870273)

[3.3. Засоби для UI 15](#_Toc483870274)

[3.4. Засоби розробки 15](#_Toc483870275)

[3. Реалізація 17](#_Toc483870276)

[3.1. Рівень даних 17](#_Toc483870277)

[3.2. Логічний рівень 22](#_Toc483870278)

[3.3. Презентаційний рівень 33](#_Toc483870279)

[4. Висновки 34](#_Toc483870280)

[4.1 Досягнені цілі 34](#_Toc483870281)

[4.2 Майбутні цілі 35](#_Toc483870282)

[5. Список літератури 36](#_Toc483870283)

# Вступ

**Тема**: розробка ігрової програми з використанням 3D графіки на мові Java.

**Мета**: ознайомитися із сферою ігрової індустрії, методиками написання сучасних відеоігор, зокрема головоломок; вивчити особливості проектування архітектури гри; закріпити набуті знання на практиці - написати ігрову програми з використанням 3D графіки на мові Java.

# Етап підготовки

На перший погляд здається неймовірним те, як швидко сьогодні ігрова індустрія набирає обертів і стає необхідним явищем в сучасному житті. Аналітики з агентства Newzoo вже встигли провести дослідження, в якому дізналися, що в цьому році ігрова індустрія принесе $108.9 млрд. Це вже на $101 млн. більше в порівнянні з 2016 роком. Сучасна ігрова індустрія багато в чому перевищує музичну. Порівняти її можна лише з індустрією кіно і телебачення.

Існує хибна думка, що ігри роблять лише для дітей. Результат опитування ж показує, що приблизно 75% цільової аудиторії ігрового бізнесу — це люди, старші за 35 років. І в кожної вікової групи є свої вподобання щодо жанрів (екшн, рольова гра, стратегія, гра-симулятор, пригоди і т.д.).

Головоломки (puzzle) можуть включатися до ігор інших жанрів як ключові елементи ігрового процесу або ж для його урізноманітнення як міні-ігри, а тому є актуальними для людей будь-якого віку. Тому ми вирішили рухатися в цьому напрямку і розробити ігру‑головоломку.

## Постановка задачі

За прототип була взята відома гра TicTacToe (хрестики-нулики). У просторах інтернету можна знайти різні її реалізації. Особливості нашої комп’ютерної програми такі:

1) 3 режими гри: «гравець проти комп’ютера», «1vs1» та багатокористувацька гра (мультиплеєр);

2) розміри поля: n х n (n = 3, 5, 6…, сторона куба), m x n (m, n > 5, m != n, сфера), n x n x n (куб);

3) збереження результатів, рейтинг;

4) можливість грати в команді;

5) елементи 3D графіки.

В режимі гри «гравець проти комп’ютера» використовуються алгоритми з різними рівнями складності: легкий, середній, складний, неможливий. На великих за розміром полях за комп’ютера грає AI.

Для багатокористувацького режиму розроблена база даних, що хоститься на сервері. Для того, щоб мати змогу грати з іншими людьми через інтернет потрібно зареєструватися. На базі результатів у мультиплеєрі ваших та інших гравців формується рейтинг.

## Прийоми при проектуванні архітектури ігор

Для якісного виконання цього завдання потрібно вивчити особливості проектування архітектури ігор та ознайомитися із методиками їх написання. В цьому ми звернулися до другого видання книги Е. Роллінгза та Д. Морріса[1] та статті на сайті it-ua.info[5].

### Компонентно-орієнтоване програмування (КОП)

Нагадаємо, що таке об’єктно-орієнтоване програмування (ООП). Це одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають три основні концепції: інкапсуляція, успадкування та поліморфізм, а об'єкт в ООП підході визначається описуваним його класом.

Проте у великих іграх архітектура є занадто складною, щоб використовувати стандартний ООП підхід. У противному випадку гарантовані постійні переробки купа коду і сильне збільшення тривалості розробки.

Проблема криється саме в успадкуванні. Зустрічаємось із проблемою тендітних базових класів — ситуація, коли змінити реалізацію типу-предка неможливо, не порушивши коректність функціонування типів-нащадків. Рішенням цієї проблеми став компонентно-орієнтований підхід (КОП).

Коротенько, суть КОП наступна: є клас-контейнер, а також клас-компонент, який можна додати в клас-контейнер, а сам об'єкт складається із контейнера і компонентів в цьому контейнері.

Компоненти нагадують інтерфейси. Але на відміну від них, компоненти мають можливість винести загальну реалізація класів окремо, а інтерфейси всього лише надають класам загальну сигнатуру функцій і властивостей.

В КОП підході об'єкт визначається компонентами, з яких він складається. Не важливо, що це за об'єкт. Важливо, що у нього є і що він вміє робити.

КОП спрощує повторне використання написаного коду — використання одного компонента в різних об'єктах. Також з різних комбінацій вже існуючих компонентів, можна зібрати новий тип об'єкта.

В нашій програмі ми вирішили використати КОП підхід при утворення об’єкта «гральний стіл», а саме: об’єкт-контейнером є сам гральний стіл; об’єктами-компонентами є команди, що в свою чергу складаються із гравців (людина або комп’ютер). Також об’єктами-компонентами для грального столу є ігрові поля. Їх кількість та сама їх структура не є наперед визначеною.

Таким чином ми можемо отримати різні об’єкти-контейнери (ігровий стіл), якщо наповнимо його різними об’єктами-компонентами (команда, ігрове поле). В свою ж чергу команда – це також об’єкт-контейнер, що складається із об’єктів-компонентів – гравців.

### Машина станів та дерево поведінки

КОП підхід – це, звичайно, дуже добре. Є в нас велика кількість як об’єктів-контейнерів так і об’єктів-компонентів, але потрібно ще організувати взаємодію між ними. Для спрощення роботи з об'єктами зі складною поведінкою використовуються машини станів, дерева поводжень.

Машина станів – це коли вся логіка об'єкта розбивається на стани, події, переходи, а також може розбиватися ще й на дії. Для спрощення написання AI в головоломках використовувати такий підхід не рекомендується, натомість рекомендують використовуються дерева поведінки (Behavior Tree).

Дерево поведінки — це деревовидна структура, в якості вузлів якої виступають невеликі блоки ігрової логіки. Кожен вузол повертає результат, від якого залежить, як будуть оброблятися інші вузли дерева.

Наприклад, у нашій грі від статусу ігрового поля в заданий момент залежить те, чи може користувач робити хід в цей момент, чи він змушений чекати ходу суперника.

Ще одним застосуванням дерева рішень є саме написання алгоритмів для гри проти комп’ютера, де вузлами дерева поведінки є клітинки поля, а результатом, який вони повертають є колір, якими вони замальовані, і чи замальовані взагалі.

## Шаблони програмування

Шаблони програмування – це напрацьовані ефективні підходи, техніки та правила вирішення задач при створенні програмного забезпечення. Вони не прив’язуються до певної мови програмування і можуть бути застосованими в основному незалежно від конкретної мови. Для якісного написання нашої ігрової програми ми вирішили використати деякі із них, а саме: MVC, репозиторій, фабричний метод, стратегія.

### MVC

Модель–вигляд–контролер (або Модель–представлення–контролер, англ. Model-view-controller, MVC) — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.



Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

Наша програма побудована за цим шаблоном і має трирівневу структуру: базу даних (модель даних), вікна (вигляд), інфраструктуру та контроллери (модель керування).

### Репозиторій

У багатьох додатках, бізнес-логіка забезпечує доступ до даних зі сховищ даних, таких як бази даних, списки SharePoint або веб-служби. Прямий доступ до даних може привести до наступного:

* дублювання коду;
* більш високий потенціал для помилок програмування;
* слабка типізація бізнес-даних;
* складність в централізації політики, пов'язаних з даними, такі як хешування;
* нездатність легко перевірити бізнес-логіку в ізоляції від зовнішніх залежностей.

Рішенням в цій ситуації стає шаблон репозиторій. Він дає можливість відокремити логіку, яка отримує дані і відображає його в моделі об'єкта з бізнес-логіки, яка діє на моделі. Бізнес-логіка повинна бути незалежною від типу даних, який містить джерело даних шару. Наприклад, джерело даних шар може являти собою базу даних, список SharePoint або веб-служби.

Репозиторій є посередником між джерелом даних шару і бізнес-шари додатки. Він запитує джерело даних для даних, відображає дані з джерела даних для бізнес-об'єкта, і зберігається зміни в комерційній організації до джерела даних.



Сховище відокремлює бізнес-логіку від взаємодії з базовим джерелом даних або веб-службою. Поділ між даними і бізнес-ярусу, має три переваги:

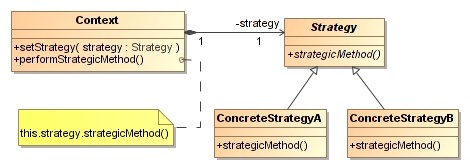
* Вона централізує логіку доступу логіки даних або веб-служби.
* Вона забезпечує точку заміщення для модульних тестів.
* Це забезпечує гнучку архітектуру, яка може бути адаптована в якості загальної конструкції еволюціонують додатки.

За допомогою цього шаблону відбувається робота з базою даних в нашій програмі. Цей підхід сильно спрощує завдання і дає можливість ізолювати рівень моделі даних від інших двох (модель представлення, модель керування).

### Стратегія

Стратегія (англ. Strategy) — шаблон проектування, належить до класу шаблонів поведінки. Відомий ще під іншою назвою - "Policy". Його суть полягає у тому, щоб створити декілька схем поведінки для одного об'єкту та винести в окремий клас.

Шаблон стратегія (Strategy) дає змогу міняти вибраний алгоритм незалежно від об'єктів-клієнтів, які його використовують.



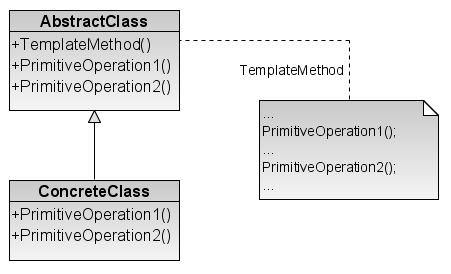
В нашій програмі використовуємо цей шаблон при побудові алгоритмів для гри в режимі «гравець проти комп’ютера». Отримаємо, що комп’ютер (об’єкт) має декілька схем поведінки, що залежить від рівня складності, обраної користувачем.

### Шаблонний метод

Шаблонний метод (англ. Template Method) - шаблон проектування, належить до класу шаблонів поведінки. Визначає основу алгоритму та дозволяє підкласам перевизначити деякі кроки алгоритму, не змінюючи структуру в цілому.

Слід використовувати шаблон шаблонний метод коли:

* треба одноразово використати інваріантні частини алгоритму, залишаючи реалізацію поведінки, що змінюється, на розсуд підкласів;
* треба відокремити та локалізувати в одному класі поведінку, що є загальною для усіх підкласів, щоб запобігти дублювання коду;
* для управління розширеннями підкласів. Можна визначити шаблонний метод таким чином, що він буде викликати операції-зачіпки у означених точках, дозволивши тим самим розширення тільки у цих точках.



В нашій програмі ми використовуємо цей шаблон для екзаменаторів ігрових полів (board checker). Поля бувають різного розміру і структури. В залежності від того, яке поле перевіряє екзаменатор, міняються його «примітивні» функції, хоча шаблонний метод не перевизначається.

# Огляд використаних технологій

## Засоби для бази даних

SQL – декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації тощо.

PostgreSQL — це об’єктно-реляційна система керування базами даних з відкритим вихідних кодом. Є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite).

ElephantSQL — це сервіс для хостингу PostgreSQL баз даних. ElephantSQL бере на себе управління такими завданнями PostgreSQL, як: інсталяція, оновлення до останньої версії та резервне копіювання.

## Засоби для інфраструктури та контролерів

Java — об’єктно-орієнтована мова програмування, основною перевагою якої є її платформо-незалежність. Вона надає різноманітні бібліотеки для створення веб-сервісів.

Spring Framework — це програмний каркас та контейнер з підтримкою інверсії керування для платформи Java. Використані такі її компоненти:

* Spring Core — основний компонент.
* Spring Data JPA — підтримує роботу з базами даних.
* Spring Boot — підтримує програмування за угодою, чим значно спрощує розробку програм для Spring.

Hibernate — засіб відображення між об’єктами та реляційними структурами, який використовує Spring для роботи з реляційними базами даних.

## Засоби для UI

JavaFX — платформа на основі Java для створення програм з насиченим графічним інтерфейсом. Може використовуватися як для створення комп’ютерних програм, що запускаються з-під операційних систем безпосередньо, для інтернет-застосунків (RIA), так і для мобільних додатків.

FXML (FX Markup Language) – мова розмітки, створена Oracle Corporation, що базується на XML, і забезпечує можливість для написання користувача інтерфейсу (UI) окремо від логіки програми.

CSS (Cascading Style Sheets) – мова, яку використовують для опису веб‑сторінок, написаних мовами розмітки.

Bootstrap – набір інструментів, який містить шаблони CSS та HTML для відображення компонентів інтерфейсу. Спрощує розробку стилів.

## Засоби розробки

IntelliJ IDEA – інтегроване середовище розробки для різних мов програмування, зокрема Java, Python, JavaScript, TypeScript та інших. Розроблене компанією JetBrains.

ERDPlus – веб-сервіс для створення ER‑діаграм моделей баз даних з можливістю експортувати модель у вигляді XML та SQL‑коду.

Pony ORM – веб-сервіс для створення ER‑діаграм моделей баз даних з можливістю експортувати модель у вигляді SQL‑коду.

Git – розподілена система контролю версій та спільної роботи. Надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок.

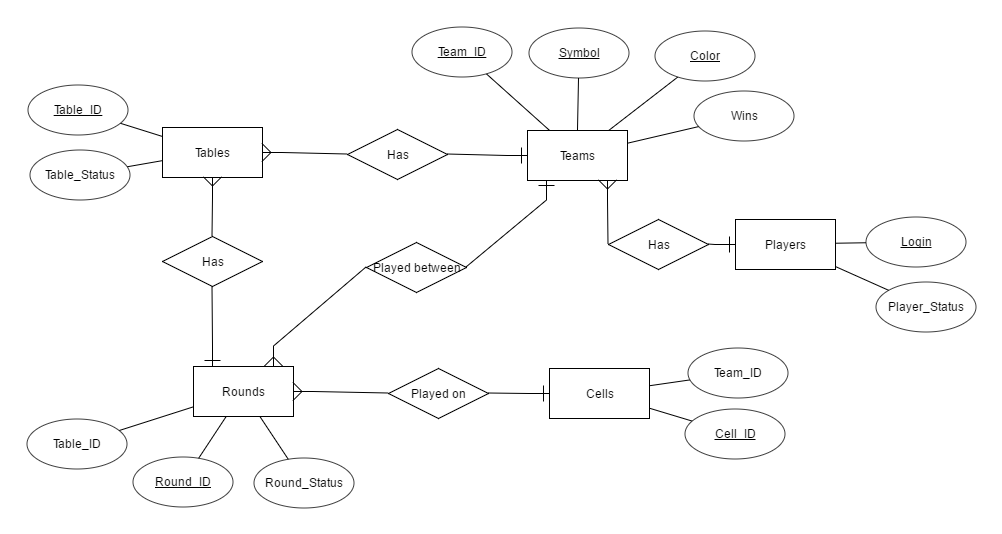
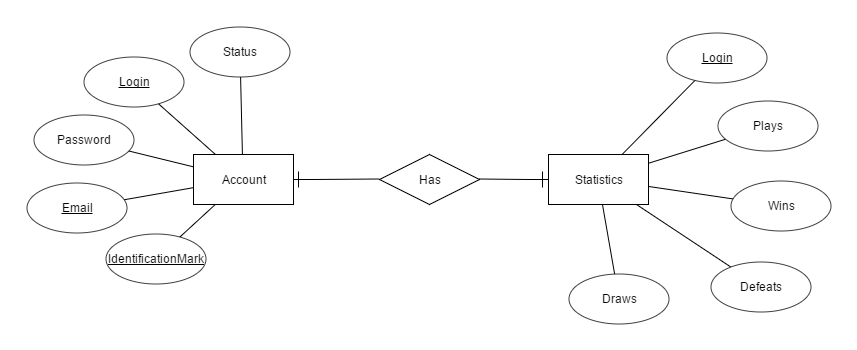
GitHub – один з найбільших веб-сервісів для спільної розробки програмного забезпечення, який базується на системі керування версіями Git.

# Реалізація

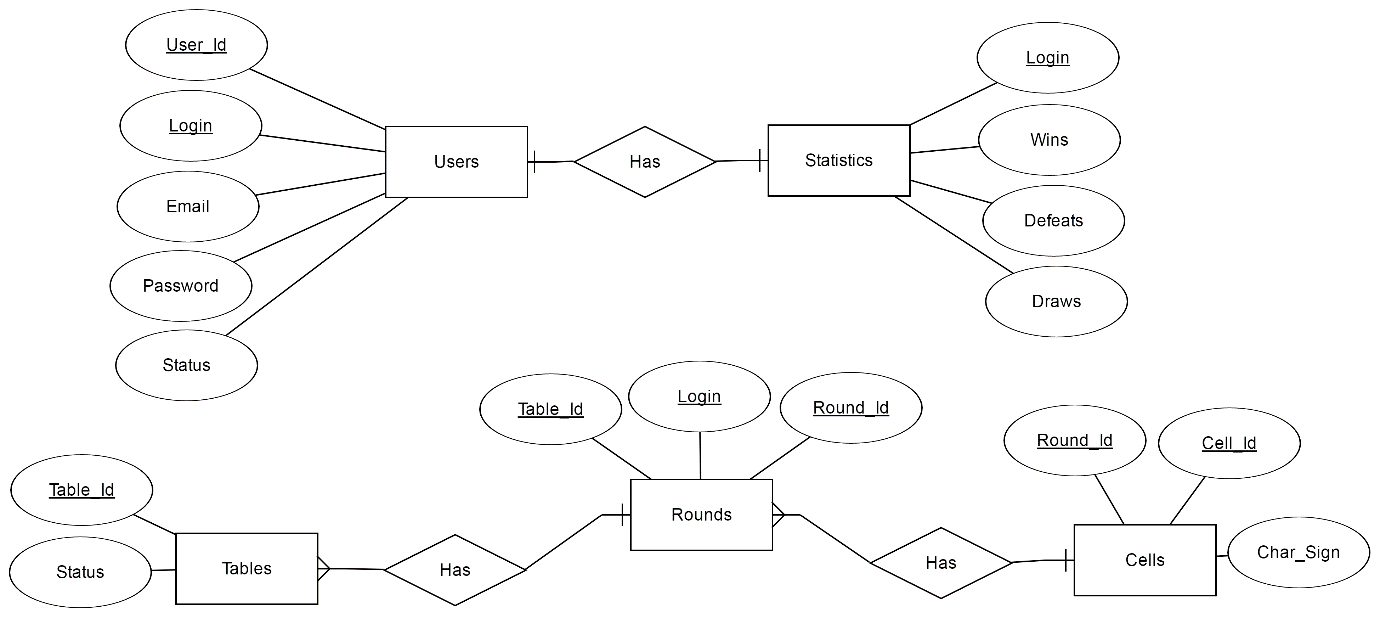
Наш проект має трирівневу архітектуру: рівень даних (БД), логічний рівень (інфраструктура, контролери, сервіси), презентаційний рівень (stages, scenes).

## Рівень даних

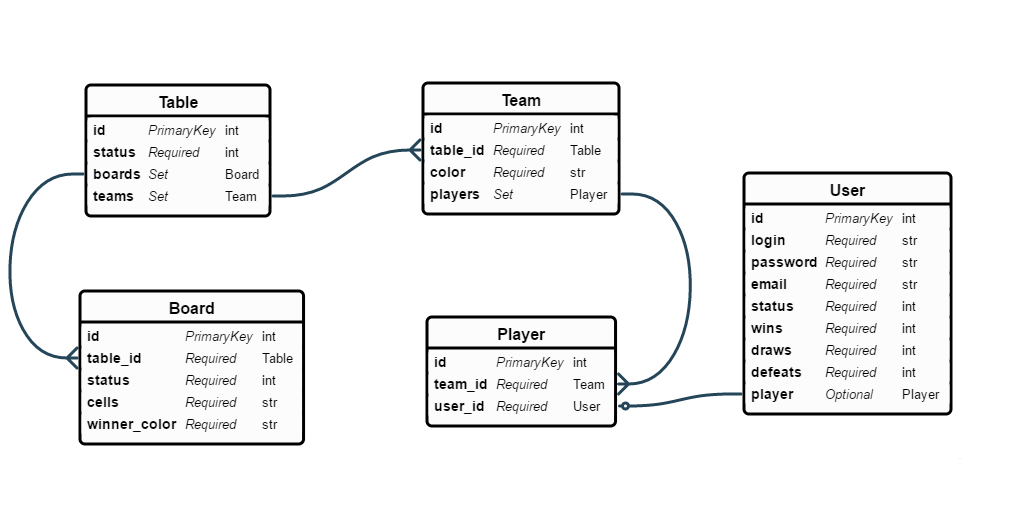
В процесі розробки модель нашої бази даних зазнавала значних змін.Мал. 1. ER‑діаграми першої версії моделей баз даних



(верхня – локальна, нижня – глобальна)

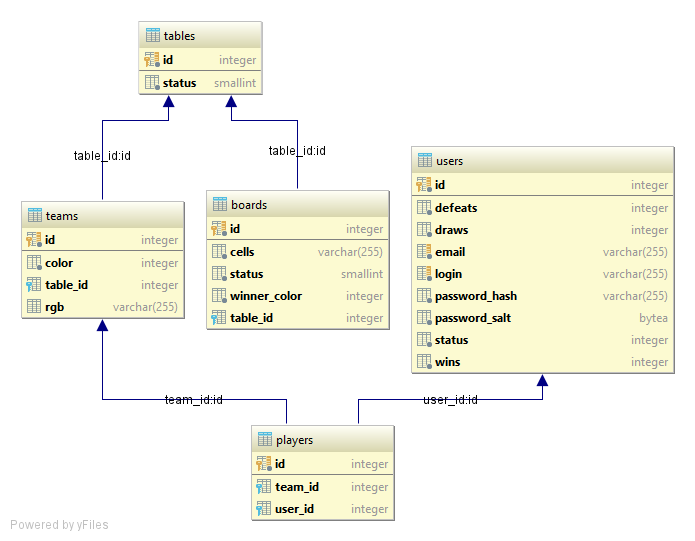


Мал. 2. ER‑діаграма другої версії моделі бази даних



Мал. 3. ER‑діаграма третьої версії моделі бази даних

Сьогодні ж модель нашої БД має такий вигляд:



Мал. 4. ER‑діаграма фінальної версії моделі бази даних

Сутності:

* користувач (User).

Атрибути:

* + - логін (login);
    - хеш-пароль (passwordHash);
    - сіль (passwordSalt);
    - почтова скринька (email);
    - статус (status) : в мережі (online), не в мережі (offline), в грі (playing), в очікуванні на початок/закінчення гри (waiting);
    - кількість перемог (wins);
    - кількість нічиїх (draws);
    - кількість поразок (defeats);
    - список гравців, за яких грав цей користувач (players).
* стіл (GameTable) – виступає в ролі турніру.

Атрибути:

* + - статус (status): створений (created), відбувається гра (active), гра завершена (finished);
    - список полів, на яких відбувається гра (boards);
    - список команд, що грають на столі (teams).
* поле (Board) – виступає в ролі раунду.

Атрибути:

* + - статус (status): створена (created), відбувається гра (active), гра завершена (finished);
    - комірки (cells);
    - переможний колір (winner\_color);
    - ідентифікатор столу, до якого належить це гральне поле (table\_id).
* команда (Team).

Атрибути:

* колір (color) – характеризує символ, який записуватиметься в комірку, на яку ходитиме кожен гравець з цієї команди;
* колір (rgb);
* список гравців, що належать до цієї команди (players);
* ідентифікатор столу, на якому грає ця команда (table\_id).
* гравець (Player).

Атрибути:

* ідентифікатор команди, за яку грає цей гравець (team\_id).
* ідентифікатор користувача (user\_id).

Примітки:

* + в кожної сутності також є атрибут id, який є первинним ключем;
  + після завершення гри, тобто коли статус певного столу стає finished, жодна із сутностей, якою ми оперували під час гри, з бази даних не видаляється.

Умовні позначення:

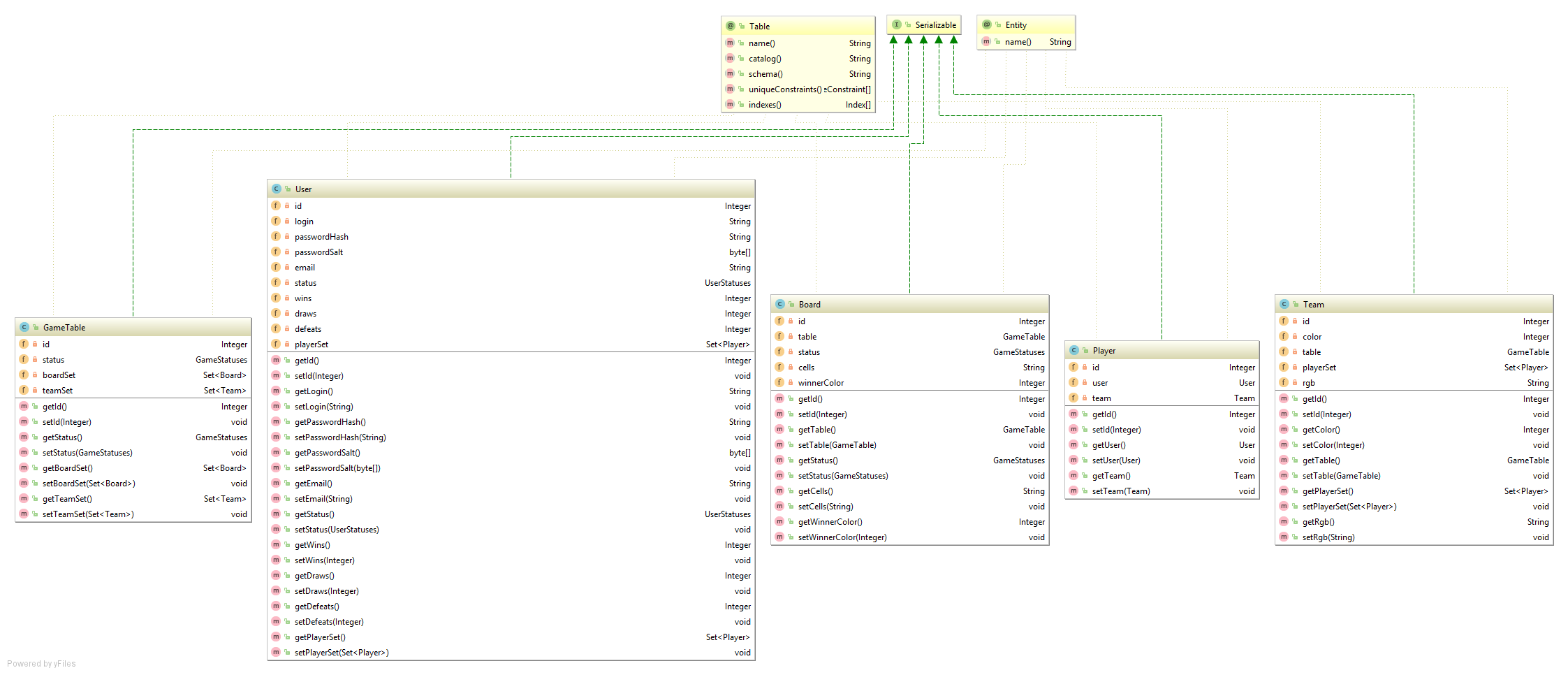
* + PrimaryKey – первинний ключ;
  + Required – обов’язковий атрибут;
  + Optional – необов’язковий атрибут;
  + Set – множинний атрибут.

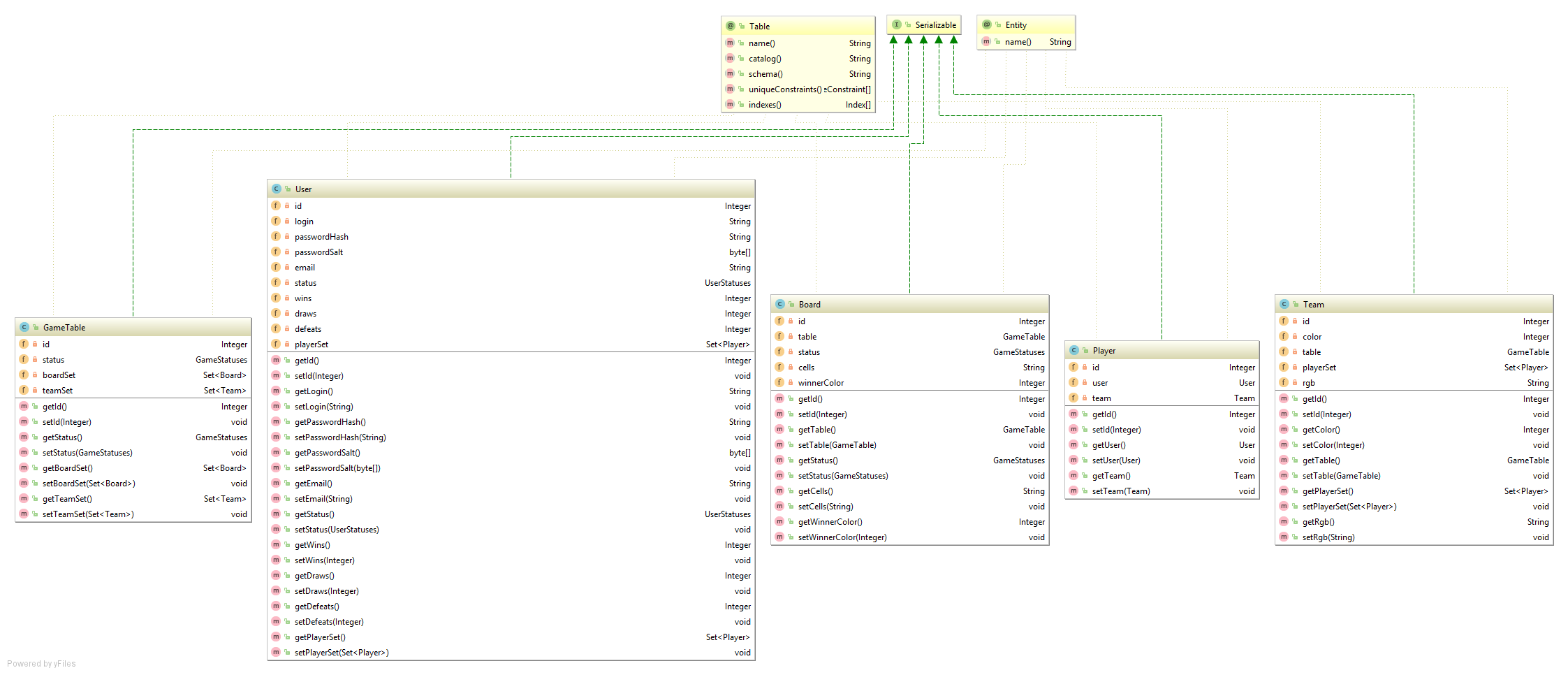
## Логічний рівень

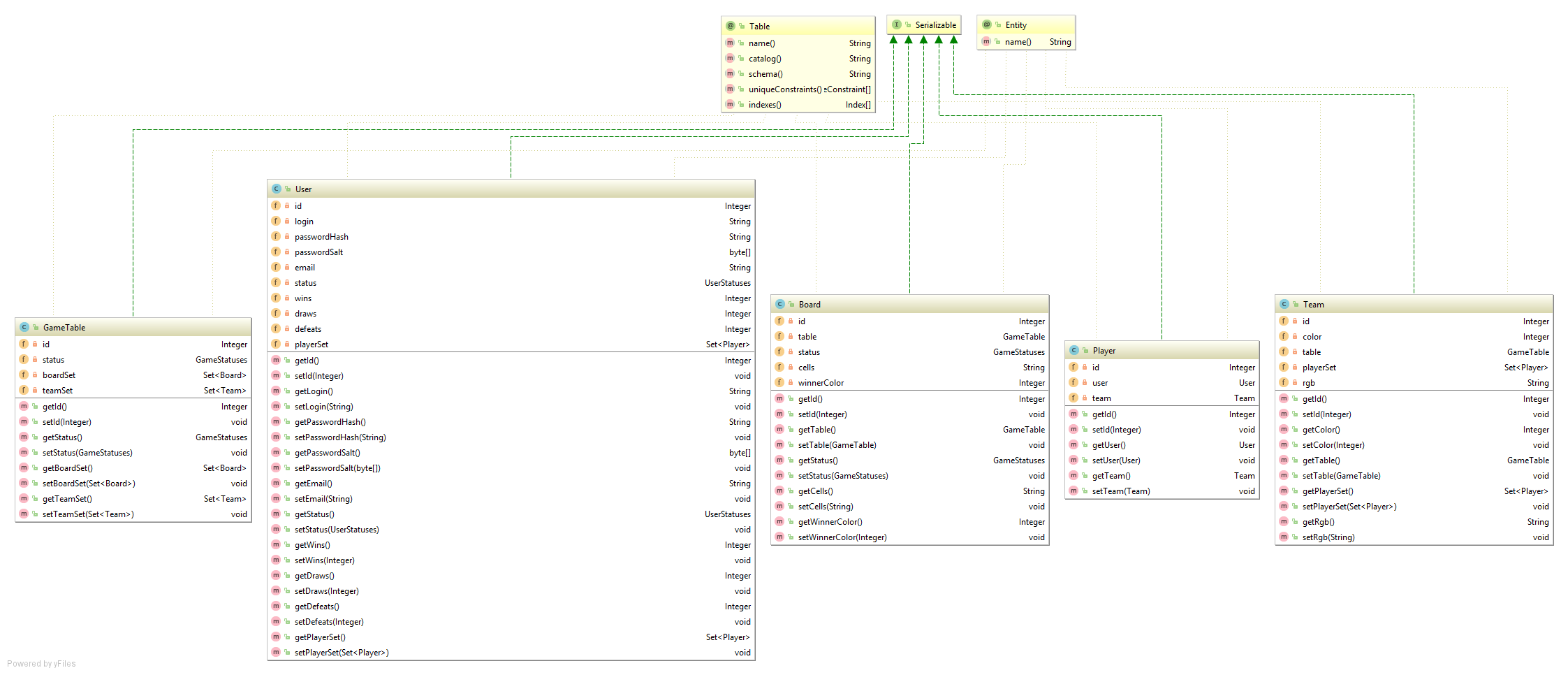
Левину долю часу було витрачено на розробку логічного рівня архітектури програми, щоб надати йому гнучкості та можливість легко розширити функціонал.

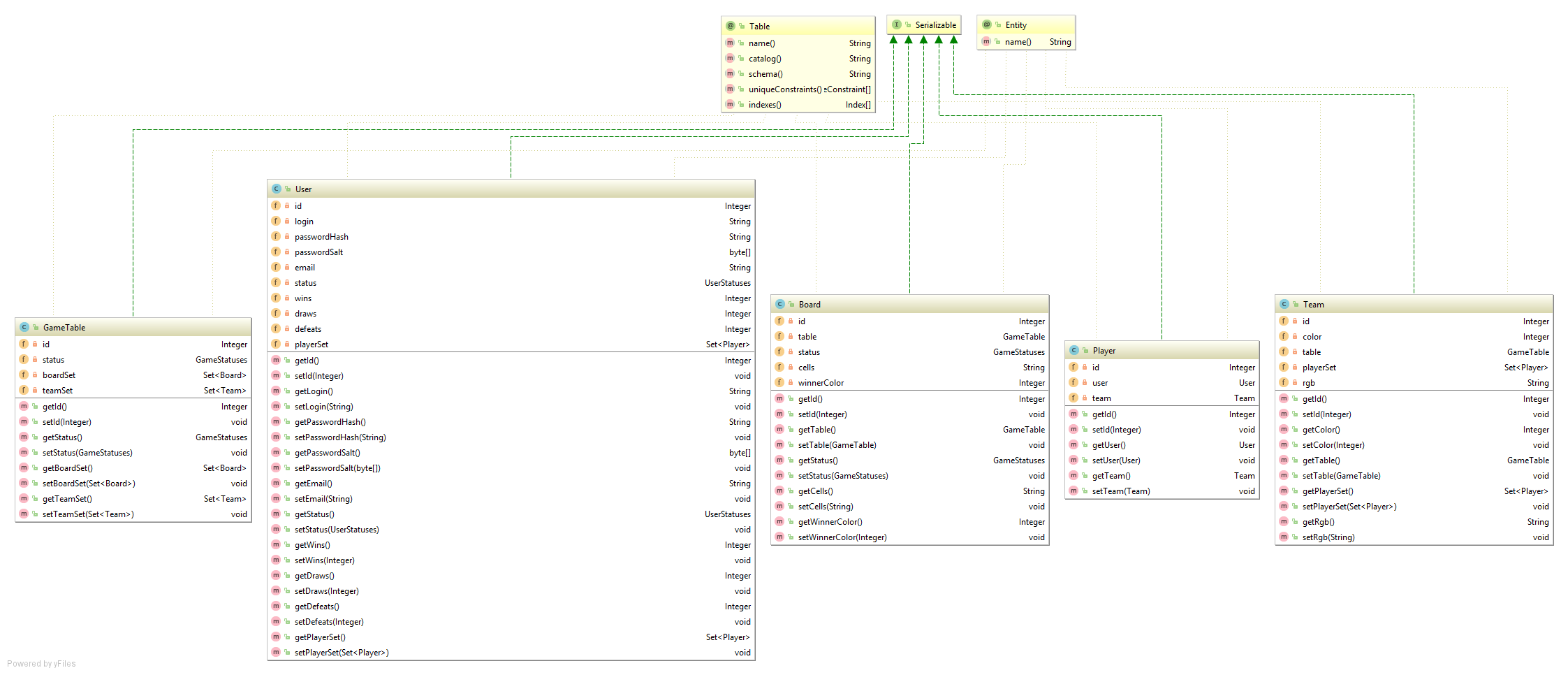
Сутності:

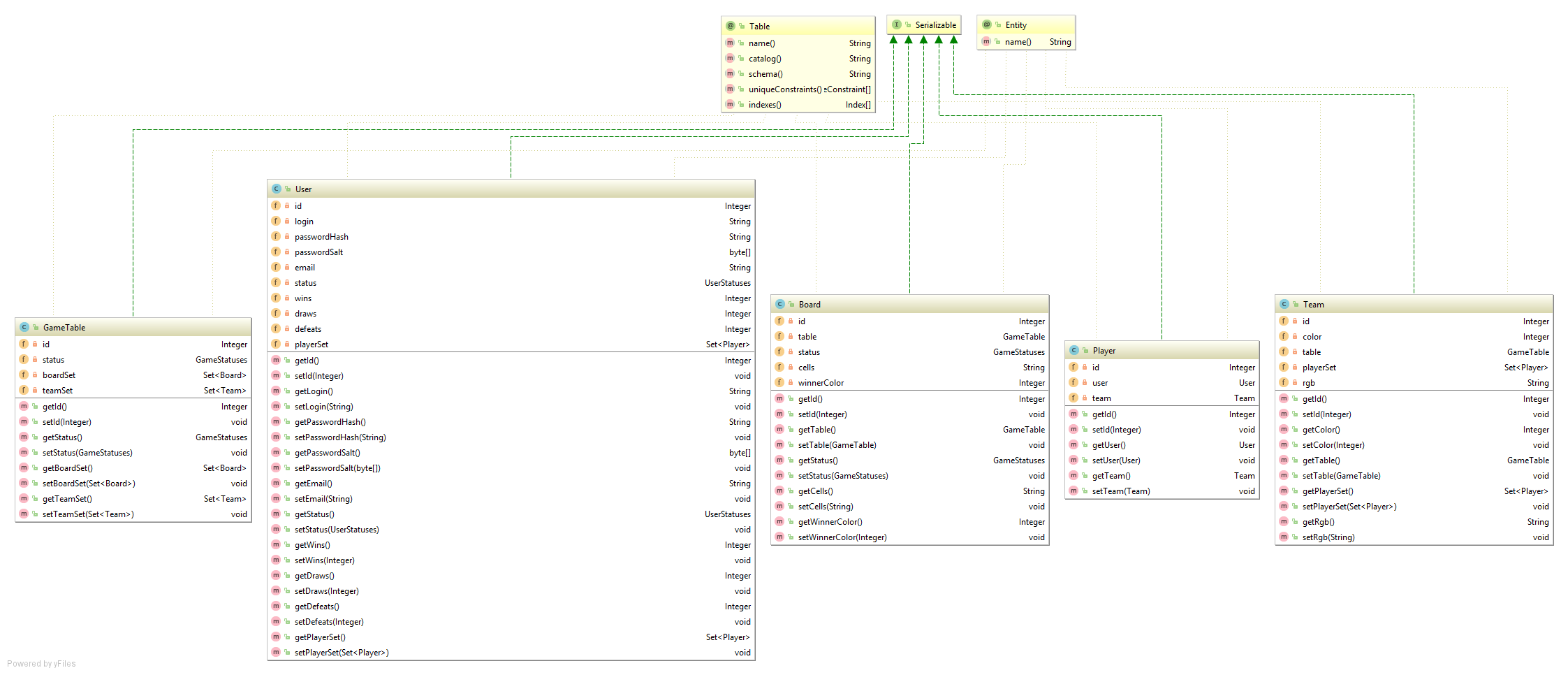
Усі сутності відповідають таблицям в базі даних.

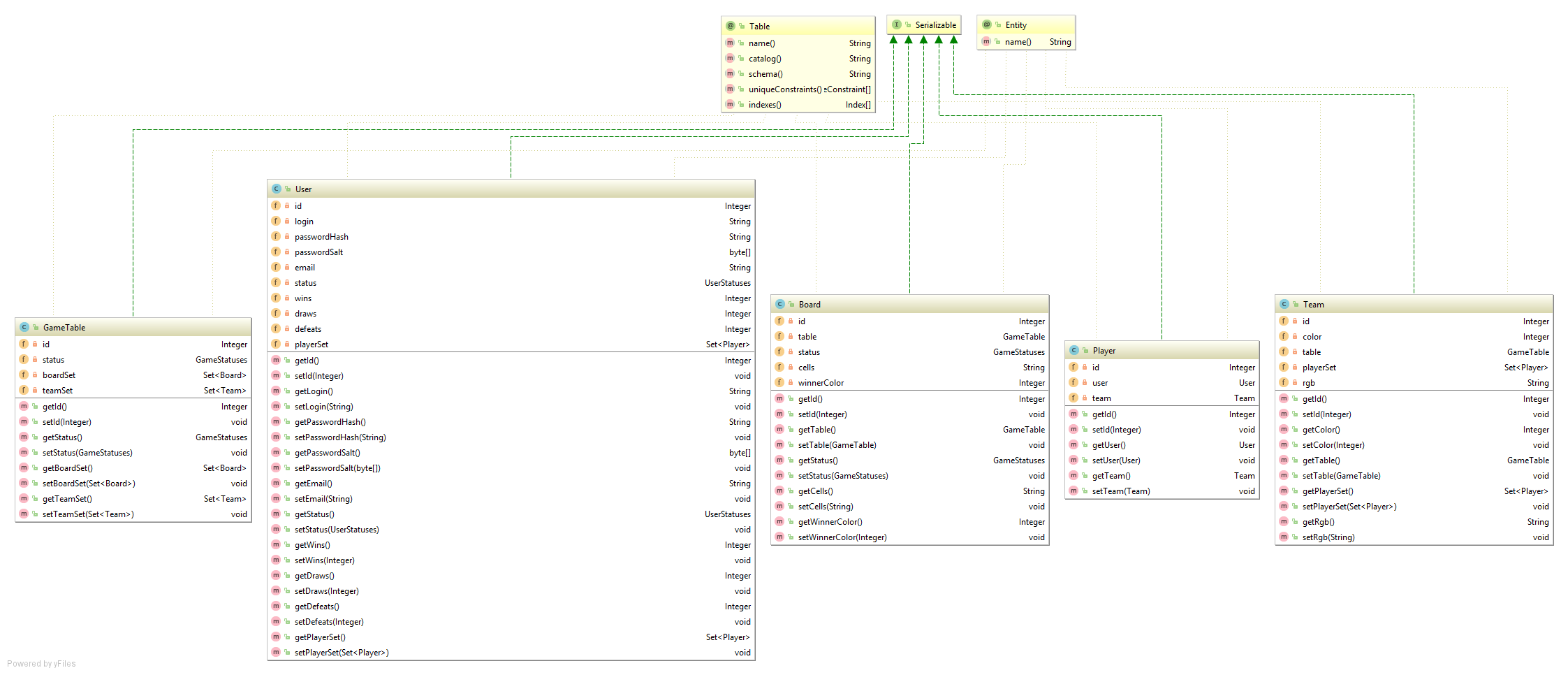








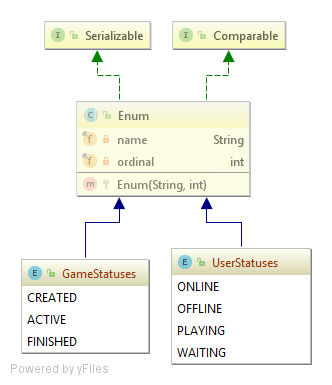




Мал. 5. Вміст domain.models

Додаткові типи:

* GameStatus – перелік, який має значення CREATED, ACTIVE та FINISHED.
* UserStatus – перелік, який має значення ONLINE, OFFLINE, PLAYING та WAITING.



Мал. 5. Вміст domain.models.statuses

Репозиторії:

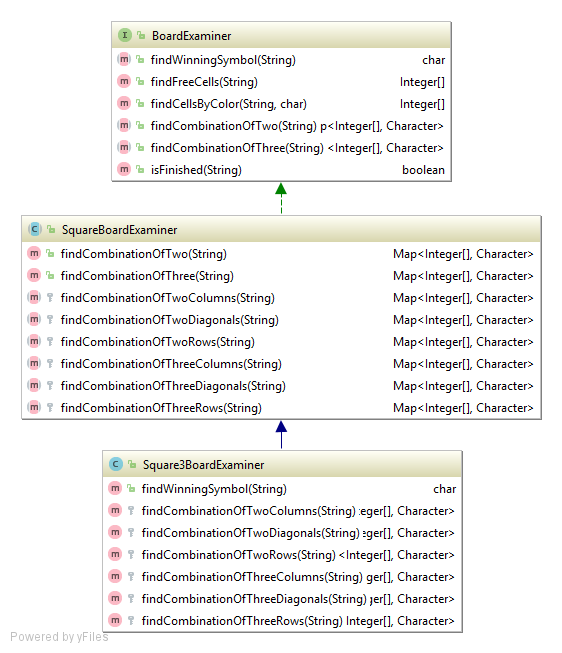
* BoardRepository – репозиторій для роботи із сутностями Board;
* GameTableRepository – репозиторій для роботи із сутностями GameTable;
* PlayerRepository – репозиторій для роботи із сутностями Player;
* TeamRepository - репозиторій для роботи із сутностями Team;
* UserRepository – репозиторій для роботи із сутностями User. Має такі методи:
  + User findByLogin(String login) – знайти користувача за заданим логіном;
  + User findByLoginAndPasswordHash(String login, String passwordHash) – знайти користувача за заданим логіном та хеш-паролем.

Примітки:

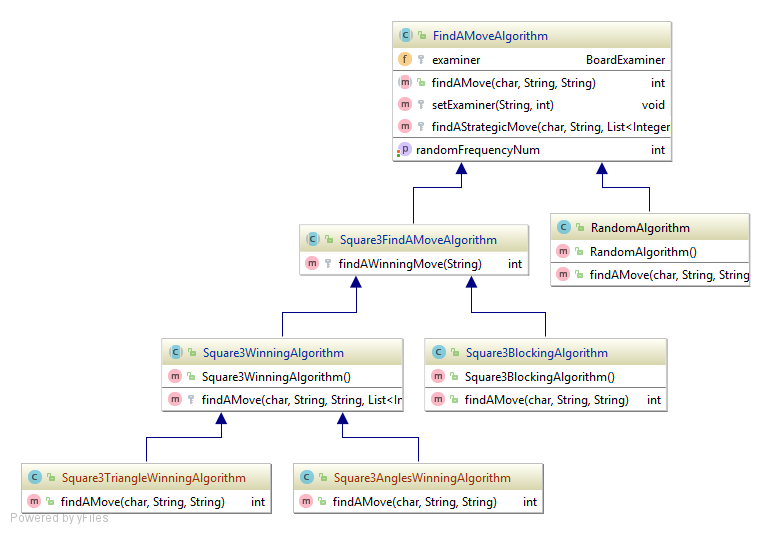
* Кожен репозиторій є інтерфейсом. Відповідні класи, які реалізують ці інтерфейси, створюються автоматично засобами Spring Boot.
* Кожен репозиторій наслідується від CrudRepository, який містить набір стандартних методів для роботи з базами даних.
* Кожен репозиторій, згенерований Spring Boot, який відповідає за зв’язок об’єктів з базою даних.

Інфраструктура:

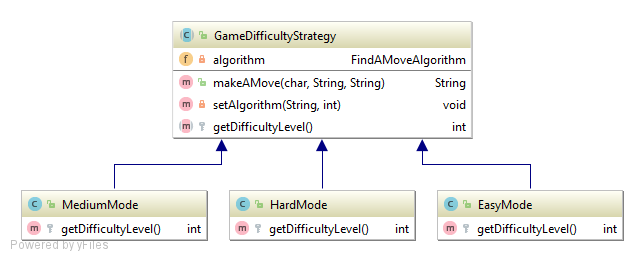
* екзаменатори дошок (board examiners):



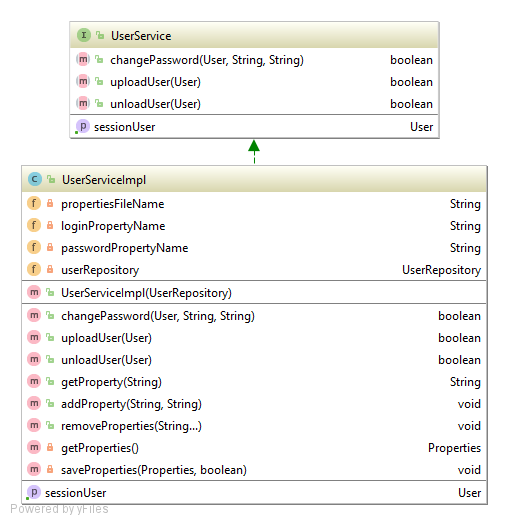
* алгоритми для знаходження ходу (algorithms):



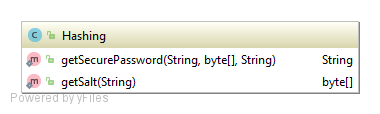
* стратегії (strategies):



* сервіси (services):

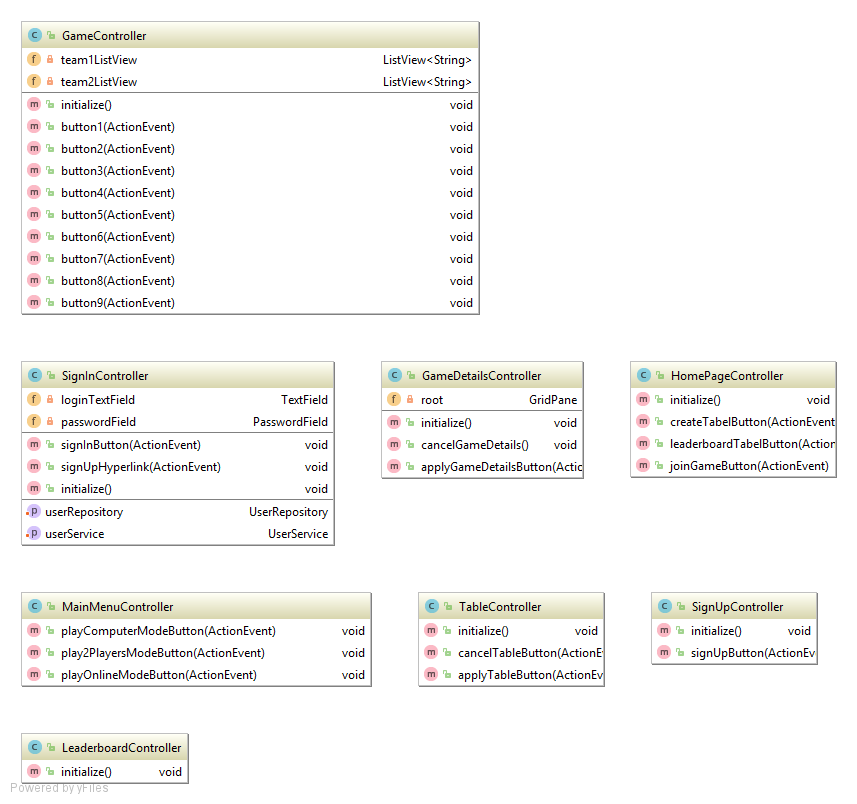


* безпека (security):

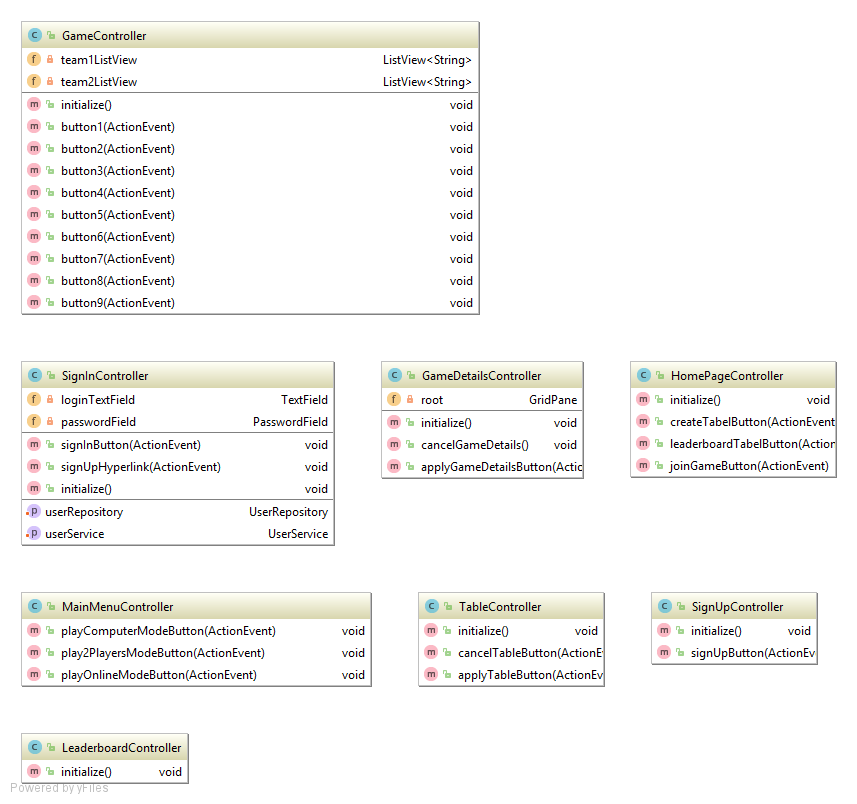


Контролери:

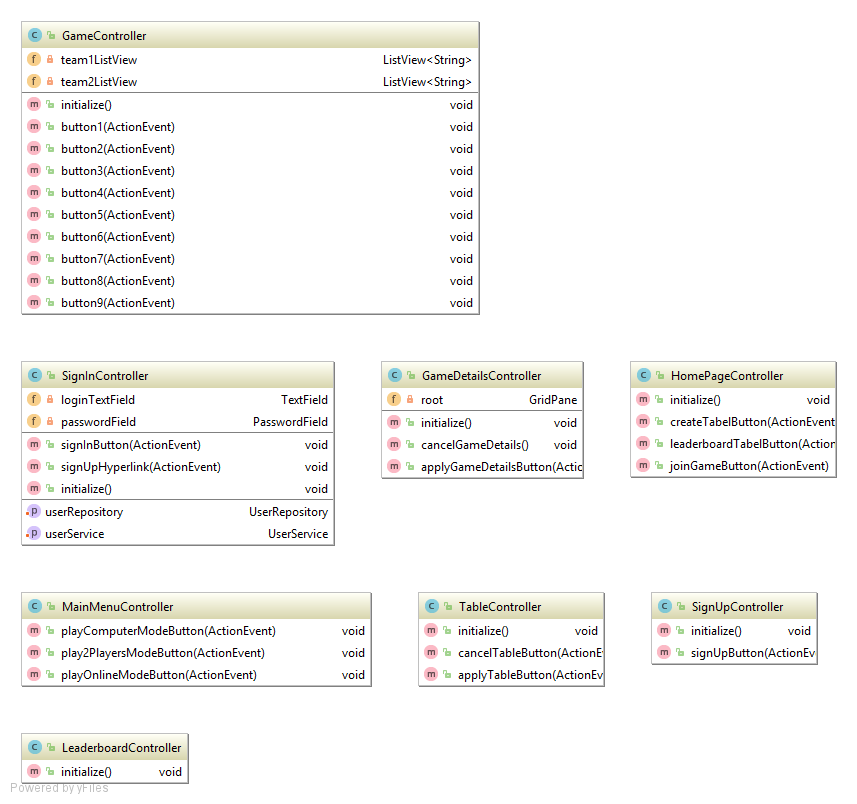
* MainMenuController:



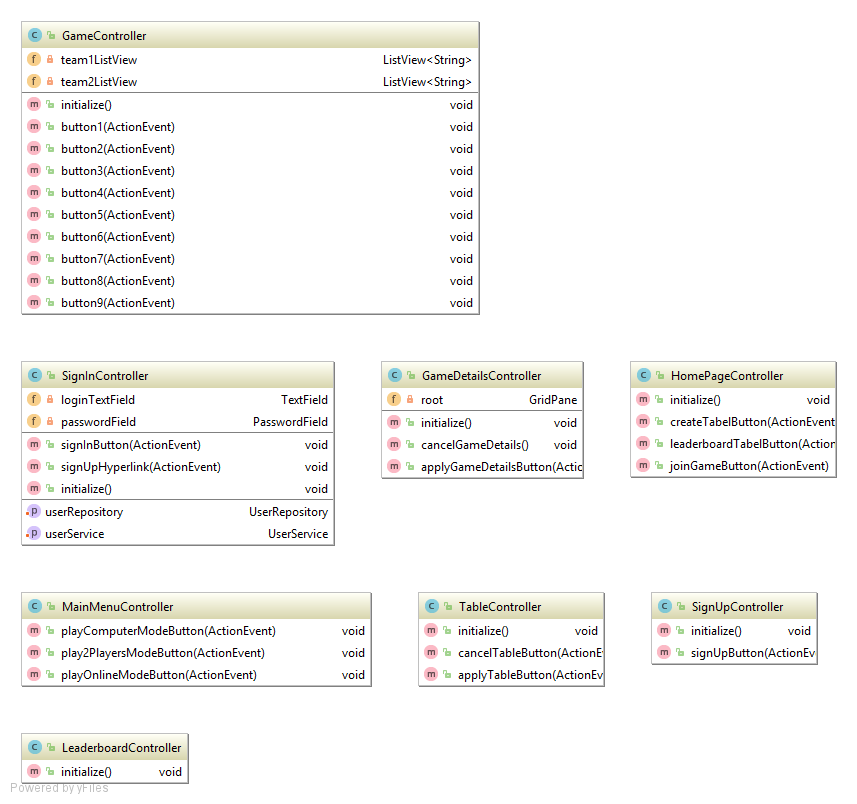
* GameDetailsController:



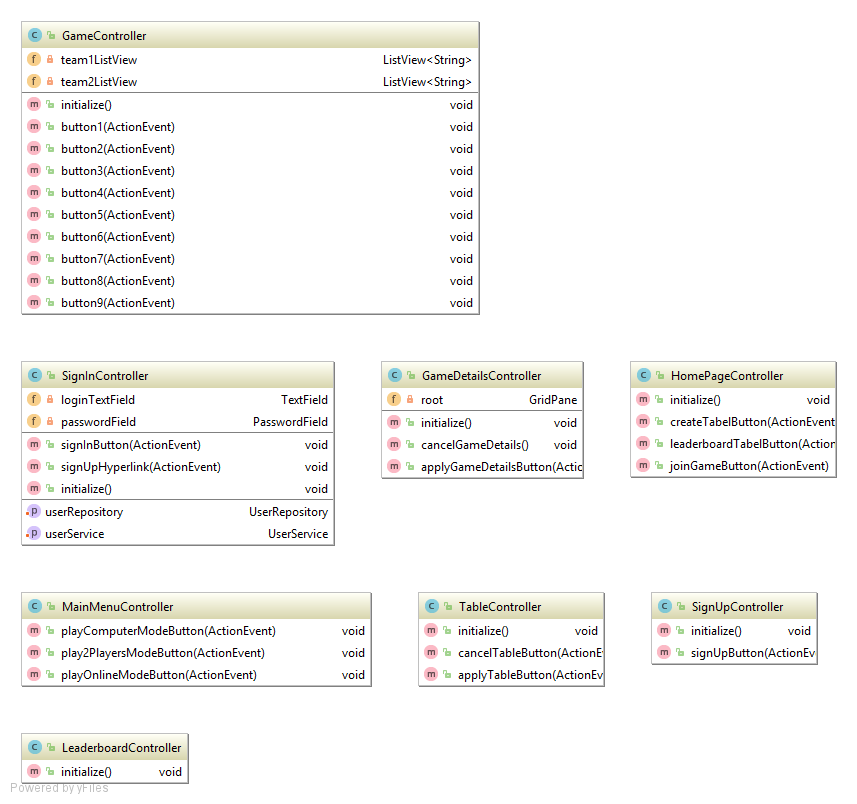
* GameController:



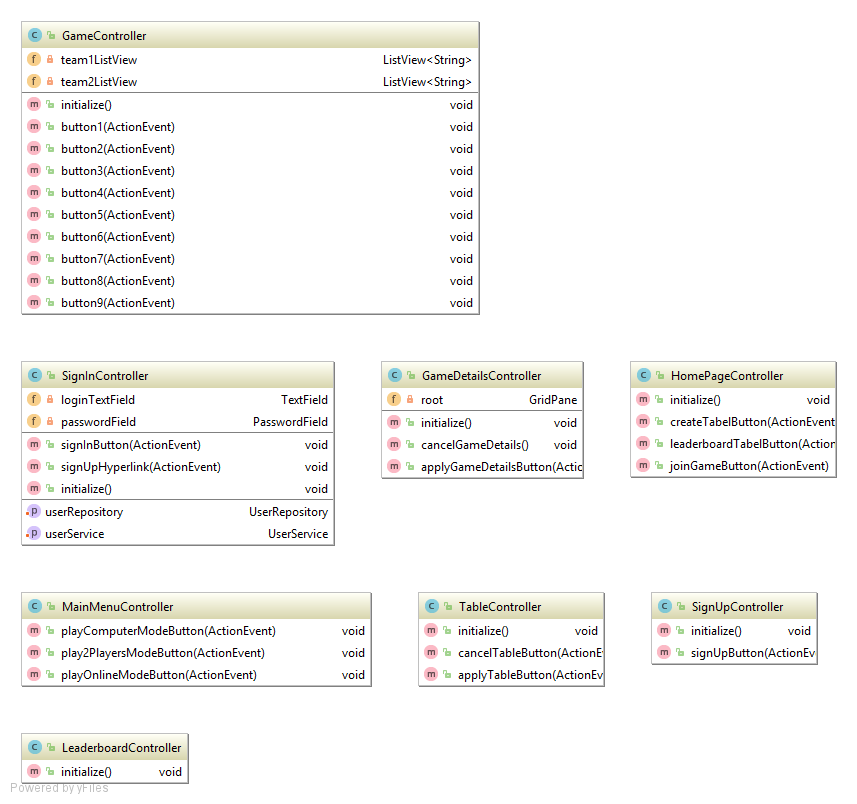
* SignInController:



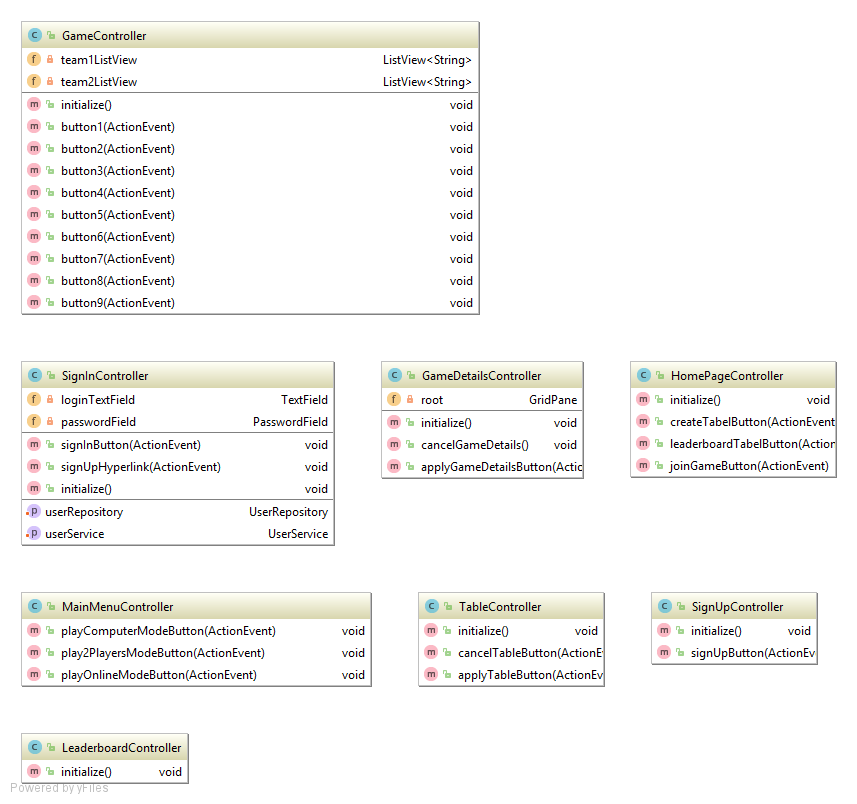
* SignUpController:



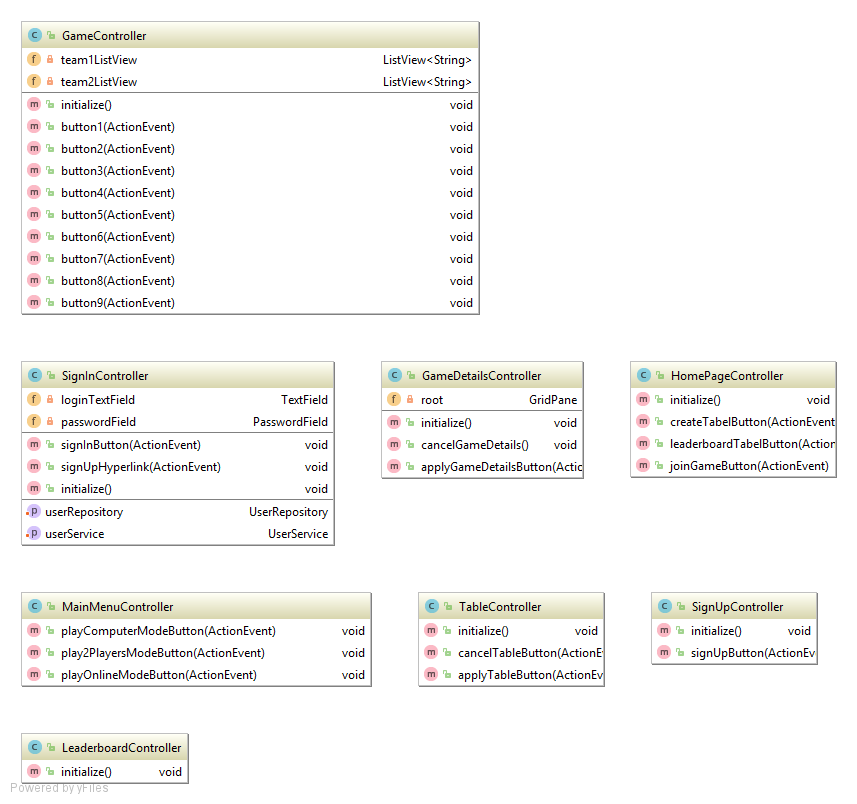
* HomePageController:



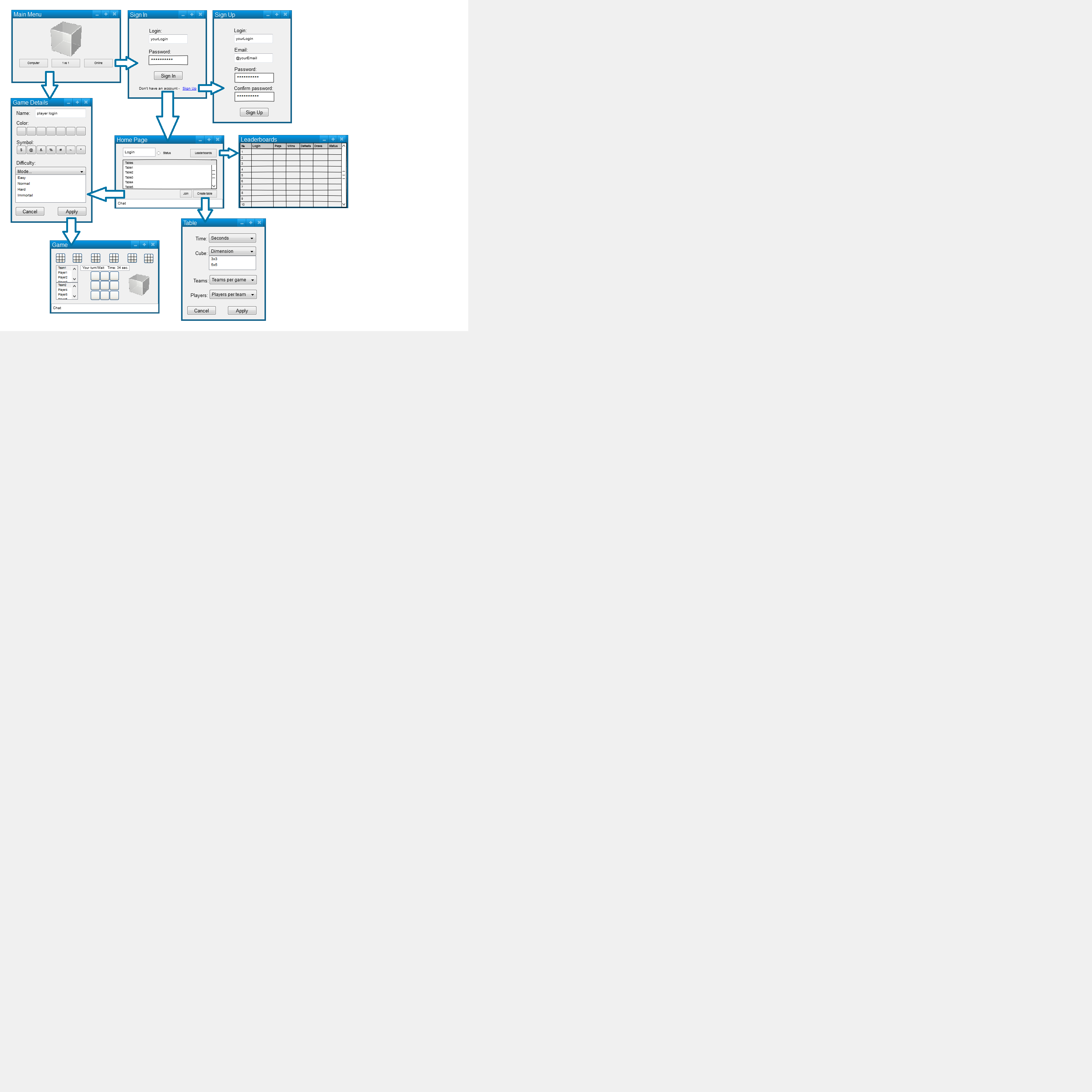
* TableController:



* LeaderboardController:



## Презентаційний рівень



# Висновки

Під час роботи над цією темою ми ознайомилися із сферою ігрової індустрії та вивчили ринок цього напрямку. Також ми опрацювали джерела про методики написання сучасних відеоігор та про особливості проектування архітектури комп’ютерної гри. Набуті теоретичні знання закріпили, написавши ігрову програму з використанням 3D графіки на мові Java.

## Досягнені цілі

* спроектували гнучку трирівневу архітектуру;
* написали базу даних;
* написали код, що дав змогу «спілкуватися» (працювати) з цією базою даних;
* написали три рівня складності: легко, нормально, важко;
* написали відповідні алгоритми для трьох рівнів складності для поля розміру 3 x 3;
* написали екзаменаторів, що перевірятимуть стан ігрового поля розміру 3 x 3;
* написали UI;
* написали відповідні контролери для гри в режимах «гравець проти комп’ютера» та «1vs1»;
* написали контролери для гри в багатокористувацькому режимі (вхід, реєстрацію, домашню сторінку, рейтинг, створення столу для гри).

## Майбутні цілі

* написати контролери для гри в багатокористувацькому режимі (мультиплеєра);
* написати відповідні алгоритми для трьох рівнів складності для поля розміру 5 x 5;
* написати екзаменаторів, що перевірятимуть стан ігрового поля розміру 5 x 5;
* написати відповідні алгоритми для трьох рівнів складності для поля, що є розгорткою сфери;
* написати екзаменаторів, що перевірятимуть стан ігрового поля розміру, що є розгорткою сфери;
* написати AI (нейронні мережі);
* написати UI та відповідні контролери для гри на 3D об’єктах та в 3D об’єктах.

# Список літератури

1. Е. Роллінгз, Д. Морріс. Проектування й архітектура ігор / 2ге видання – 2006р.).
2. Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley Professional Computing Series) / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides; Addison-Wesley Professional, 1994. - 416 p.
3. Heer, J.; Agrawala, M. (2006). "Software Design Patterns for Information Visualization". IEEE Transactions on Visualization and Computer Graphics. 12 (5): 853–60.
4. "Introduction to Software Engineering/Architecture/Design Patterns – Wikibooks, open books for an open world". en.wikibooks.org. Retrieved 2015-12-26.
5. Herbert Schildt. Java: The Complete Reference, Eighth Edition
6. <http://it-ua.info/news/2015/04/13/priyomi-pri-proektuvann-arhtekturi-gor.html>.
7. <http://www.vitaliypodoba.com/2014/06/programming-patterns-intro/>
8. [https://uk.wikipedia.org/](https://uk.wikipedia.org/wiki/Модель-вид-контролер)
9. <https://msdn.microsoft.com/en-us/library/ff649690.aspx>