DOCUMENTAȚIE

Profile Matcher Service

Nicolae-Vlăduț Mărgineanu

BUCUREŞTI

2024

CUPRINS

SINOPSIS

Documentație aplicație – Profile Matcher Service

ABSTRACT

Documentation App - Profile Matcher Service

1 INTRODUCERE

Documentație aplicație – Profile Matcher Service – specificare tehnologii

2 ANALIZA ȘI SPECIFICAREA CERINȚELOR

Unul dintre procesele cheie în dezvoltarea de software este specificația software-ului, al cărui scop este să definească ce funcționalități sunt necesare produsului software.

2.1 Cerințe funcționale pentru aplicație

Serviciul de potrivire a profilului va trebui să facă următoarele:

- Pe baza ID-ului unic al clientului, ar trebui să extragă profilul actual al jucătorului din baza de date la alegere

- Pe baza campaniilor curente în desfășurare, ar trebui să actualizeze profilul actual al jucătorului (imitați un serviciu API care va returna lista campaniilor curente care rulează)

Fluxul ar trebui să fie acesta:

- Clientul va apela API-ul de potrivire a profilului aici: GET /get_client_config/{player_id}
- Serviciul va obține profilul complet din baza de date și apoi va potrivi profilul actual al jucătorului cu setările curente ale campaniei și va determina dacă profilul actual al jucătorului se potrivește cu oricare dintre condițiile campaniei (potriviri)

Dacă cerințele sunt îndeplinite, atunci profilul actual al jucătorului va fi actualizat (adăugați numele campaniei la profil – campanie activă) și returnat utilizatorului.

2.2 Cerințe nefuncționale pentru pentru aplicație

Arhitectura bazei de date a fost realizată cu ajutorul aplicației MySQL Workbench. Această aplicație oferă modelare de date, dezvoltare de SQL și instrumente de administrare pentru configurarea server-ului, administrarea utilizatorilor și copiile datelor de rezervă [1]. Aplicația oferă funcționalități precum inginerie directă și inversă și instrumente vizuale pentru crearea, executarea și optimizarea interogărilor SQL.

3 SOLUȚIA PROPUSĂ

Arhitectura aplicației web de semnare oferă o dezvoltare flexibilă prin separarea nivelurilor logice din aplicație. Arhitectura este una multi-nivel, astfel aplicația fiind ușor de extins pe viitor prin adăugarea de noi module și funcționalități.

[1] https://dev.mysql.com/doc/workbench/en/, accesat la 10 Mai 2023

3.1 Tehnologii folosite în implementare

Pentru dezvoltarea aplicației web de semnare se folosesc cele mai potrivite tehnologii pentru realizarea unei arhitecturi multi-nivel. Tehnologiile utilizate în dezvoltarea aplicației sunt:

- JDK (minimum versiunea 17) – este un mediu de dezvoltare pentru construirea de aplicații folosind limbajul de programare Java;
- Apache Maven – este un instrument pentru managementul proiectelor care automatizează proiecte Java;
- Framework-ul SpringBoot – oferă un model cuprinzător de programare și configurare pentru aplicațiile de întreprindere moderne bazate pe Java pe orice tip de platformă de implementare[2];
- Framework-ul Spring Data JPA - folosit pentru accesul la datele dintr-o bază de date;
- SQL (Structured Query Language) – limbajul de programare standardizat utilizat pentru gestionarea bazei de date relaționale;
- GIT – sistemul de versionare pentru gestionarea codului aplicației web de semnare oferit de platforma GitHub.

3.2 Arhitectura aplicației

Framework-ul SpringBoot se bazează pe o arhitectură multi-nivel.

Aplicația este împărțită în următoarele niveluri, fiecare având un scop bine definit:

- Controller – se vor primi cererile de tip HTTP de la aplicația server REST-full (sunt implementate end-point-urile pentru cererile de tip HTTP). Controller-ul va mapa cererea și o va gestiona. Dacă trebuie efectuată vreo logică pentru o funcționalitate dorită, se va apela nivelul Command;
- Command – reprezintă un bean care este folosit pentru a încapsula toata informația necesara (o lista de parametri) pentru a efectua o acțiune sau pentru a declanșa un eveniment mai târziu. De obicei se va apela nivelul Service. Nivelul are la baza Design Pattern-ul Command [3].
- Service – se va implementa logica care conduce la funcționalitățile de bază;
- Repository – sunt gestionate obiectele care realizează conexiunea la datele stocate în baza de date. Acest nivel va fi mapat cu JPA (Jakarta Persistence API) la o clasă entitate.

Pe lângă aceste niveluri, se folosesc si bean-uri auxiliare din pachetul Assembler, cu scopul de a construi un DTO după specificațiile primite.

În cele ce urmează, se vor prezenta clasele principale din nivelurile logice ale arhitecturii SpringBoot din aplicație, precum se poate observa în Figura de mai jos.

[2] https://spring.io/projects/spring-framework, accesat la 20 Mai 2023
[3] https://medium.com/codex/how-i-implemented-command-pattern-in-spring-boot-870c1f2c73b0

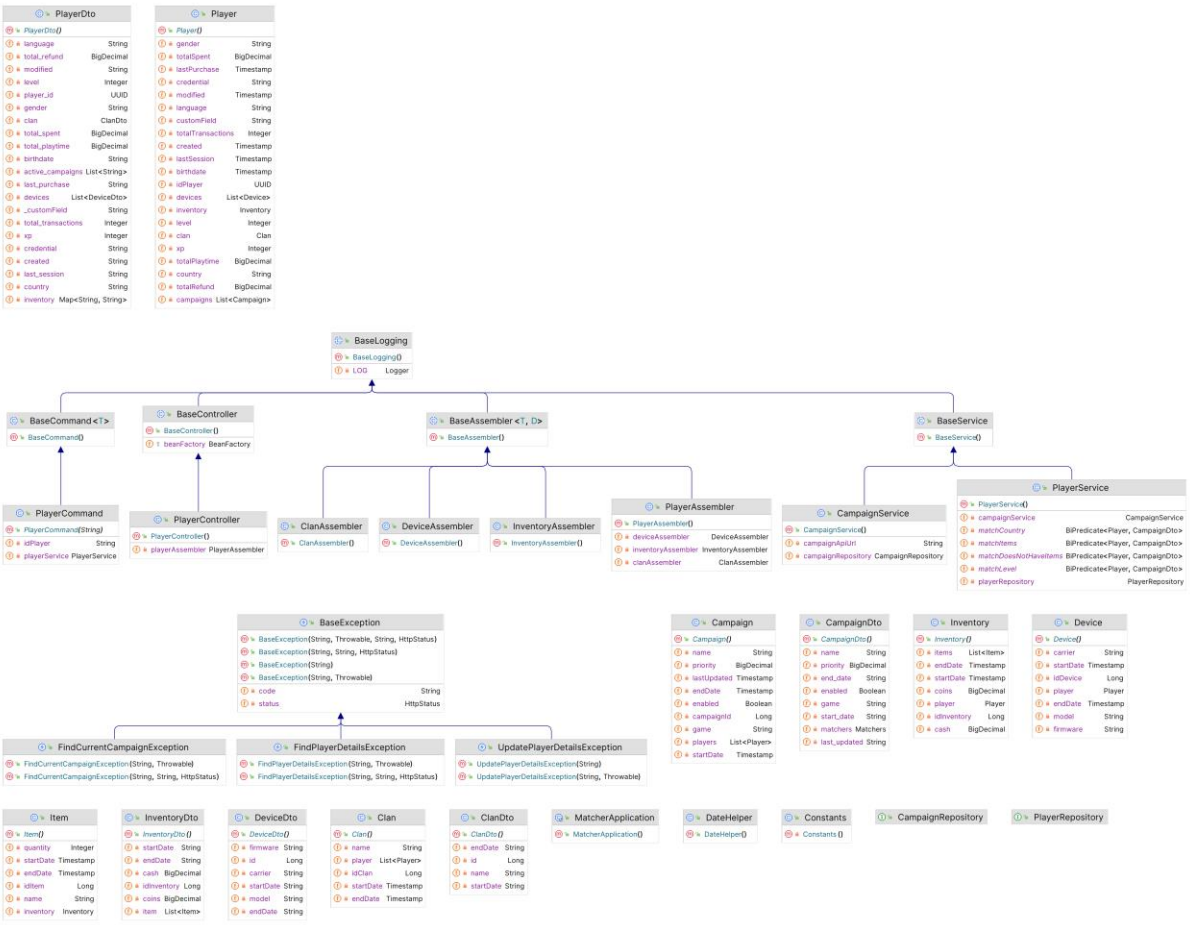


Figura 1 Diagrama claselor pentru aplicație

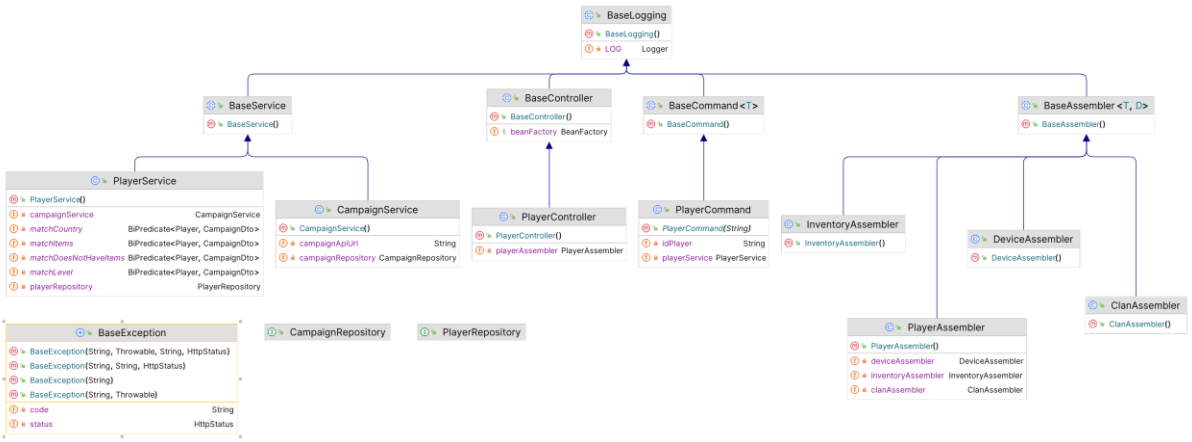


Figura 2 Diagrama claselor principale pentru aplicație

O entitate reprezintă un tabel stocat într-o bază de date. Fiecare proprietate a unei entități reprezintă un rând în tabel.

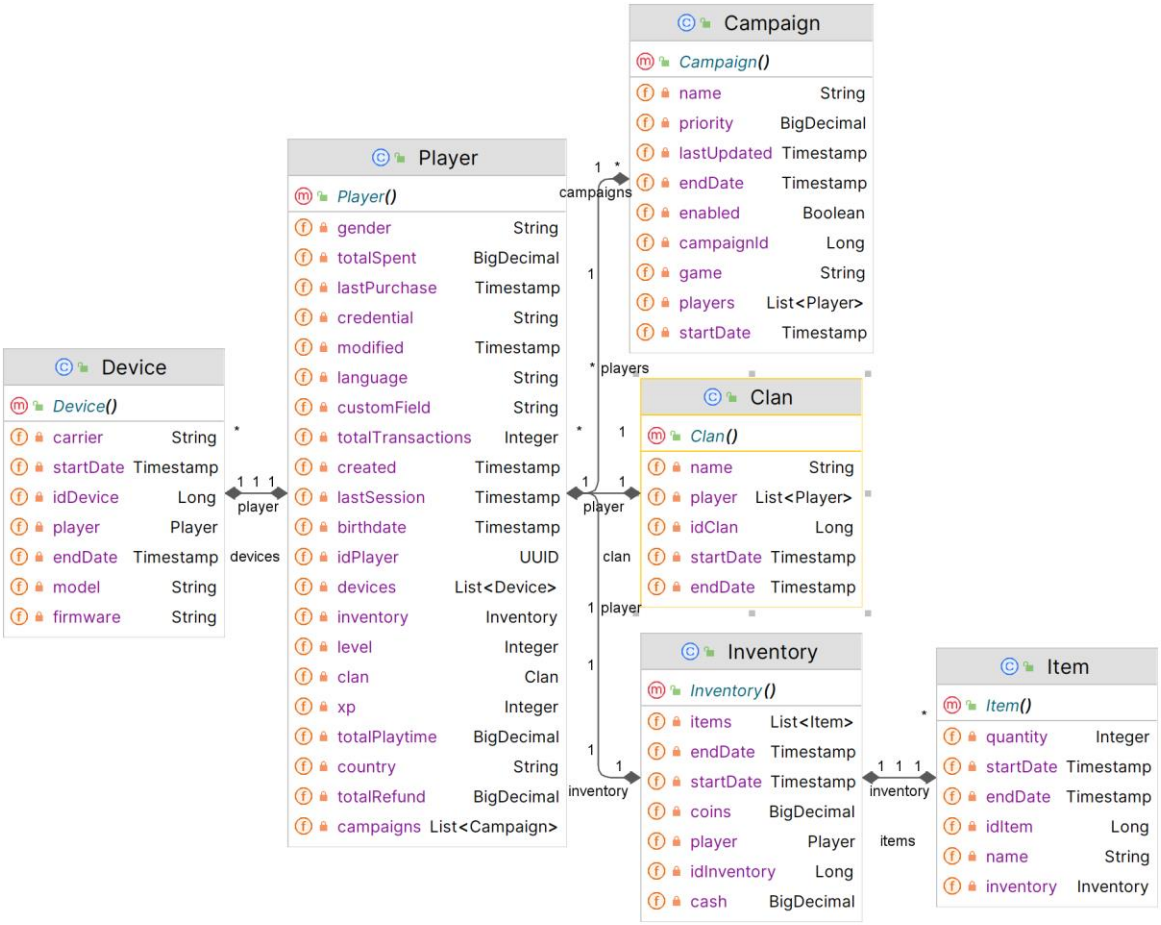


Figura 3 Diagrama de clase din pachetul entity

3.3 Arhitectura bazei de date

Arhitectura bazei de date este realizată cu un model relațional din DBMS (Database Management System) pentru a organiza și gestiona datele stocate. Astfel, se creează 6 tabele bi-dimensionale, interdependente, în care se stochează date, cunoscute și ca relații în care fiecare rând reprezintă o entitate și fiecare coloană reprezintă proprietățile entității.

În Figura de mai jos, se pot observa cele 6 tabele, cu constrângerile de integritate specifice, și anume: player, device, clan, inventory, item, campaign.

Tabela player reprezintă entitatea principală în care se vor stoca profilurile jucătorilor. Aceasta are următoarele constrângeri de integritate:

- cheia primară ID_PLAYER - pentru identificarea unică a înregistrărilor;
- cheia externă INVENTORY – tabela PLAYER se relaționează cu tabela INVENTORY pe cheia primară ID_INVENTORY;
- cheia externă CLAN – tabela PLAYER se relaționează cu tabela CLAN pe cheia primară ID_CLAN;
- NOT NULL - pentru coloanele CREDENTIAL, CREATED - înregistrările nu pot să conțină valori nule.

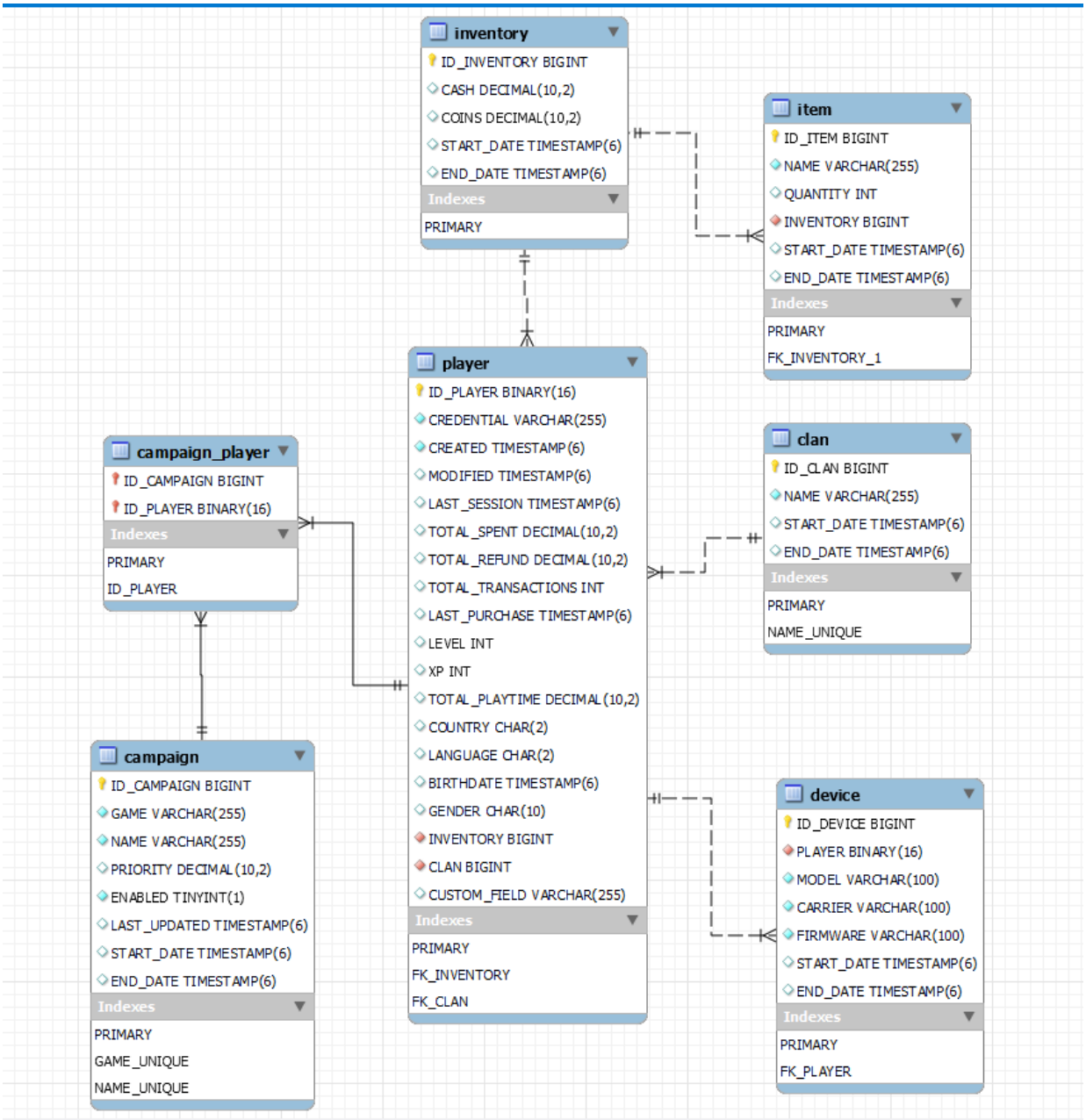


Figura 4 Diagrama model bază de date – relații entități

4 DETALII DE IMPLEMENTARE

Pe parcursul dezvoltării aplicației s-au implementat funcționalitățile precizate în capitolele de mai sus, ținând cont de arhitectura realizată. În continuare, se vor detalia aspectele importante legate de implementare, precum și algoritmii folosiți.

4.1 API Swagger

Player-controller GET: /get_client_config/{player_id}

Parameters: player_id (required)

Responses: PlayerDto


```
{
    "player_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "credential": "string",
    "created": "string",
    "modified": "string",
    "last_session": "string",
    "total_spent": 0,
    "total_refund": 0,
    "total_transactions": 0,
    "last_purchase": "string",
    "active_campaigns": [
        "string"
    ],
    "devices": [
        {
            "id": 0,
            "model": "string",
            "carrier": "string",
            "firmware": "string",
            "startDate": "string",
            "endDate": "string"
        }
    ],
    "level": 0,
    "xp": 0,
    "total_playtime": 0,
    "country": "string",
    "language": "string",
    "birthdate": "string",
    "gender": "string",
    "inventory": {
        "additionalProp1": "string",
        "additionalProp2": "string",
        "additionalProp3": "string"
    },
    "clan": {
        "id": 0,
        "name": "string",
        "startDate": "string",
        "endDate": "string"
    },
    "get_customField": "string"
```

```
}
```

4.2 Spring Data JPA

Framework-ul Spring Data este folosit pentru accesul la conținutul unei baze de date. Obiectivul de proiectare al acestui framework este de a oferi un model familiar și consistent bazat pe Spring pentru toate tehnologiile de acces la date, cum ar fi bazele de date relaționale sau non-relaționale, precum și pentru tehnologiile bazate pe cloud[4].

4.2.1 Jakarta Persistence API

În implementarea curentă, am folosit interfața JpaRepository pentru nivelul logic Repository. JpaRepository conține API-urile pentru operațiunile CRUD de bază, API-urile pentru paginare și API-urile pentru sortare.

Pentru gestionarea tranzacțiilor se folosește adnotarea @Transactional.

4.2.2 Implementarea entităților

Entitățile din Jakarta Persistence API sunt POJO-uri (Plain Old Java Object) care reprezintă date care pot fi stocate în baza de date

În procesul de implementare al entităților, se folosesc câteva adnotări importante pe care le vom prezenta mai jos:

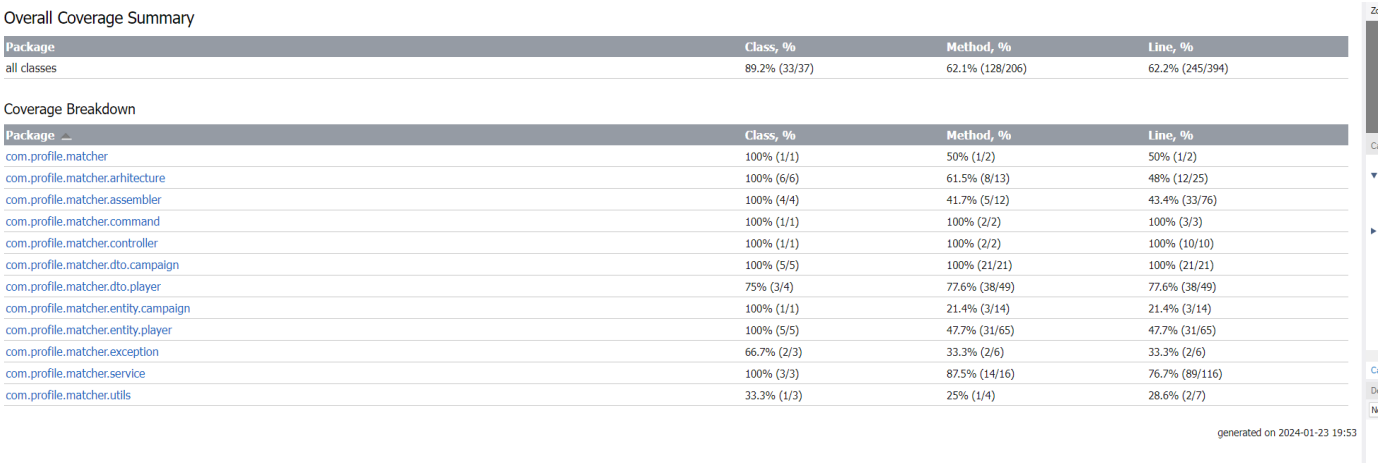
- @Entity - această adnotare permite managerului de entitate să folosească această clasă și să o pună în contextul aplicației Spring;
- @Table - asociază o clasă cu un tabel din baza de date;
- @Id - reprezintă că acest câmp este cheia primară;
- @GeneratedValue(strategy = GenerationType.IDENTITY) - definește strategia pentru generarea cheii primare;
- @Column - denotă o coloană din baza de date cu care va fi asociat acest câmp.

De asemenea, am folosit biblioteca Lombok bazată pe adnotări care ne permite să reducem codul standard. Am folosit adnotările @Getter și @Setter, în care biblioteca Lombok generează automat getter-ul și/sau setter-ul implicit. Implementarea implicită pentru getters se ocupă doar de returnarea câmpului adnotat. De asemenea, am folosit adnotarea @NoArgsConstructor, în care Lombok generază automat un constructor fără parametri, o condiție necesară pentru clasele de tip entitate în cazul nostru.

[4] https://www.geeksforgeeks.org/introduction-to-the-spring-data-framework/, accesat la 5 Mai 2023

5 EVALUAREA REZULTATELOR

Pentru procesul de testare s-au folosit teste unitare – Junit 5 impreuna cu biblioteca Mockito.



Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	89.2% (33/37)	62.1% (128/206)	62.2% (245/394)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
com.profile.matcher	100% (1/1)	50% (1/2)	50% (1/2)
com.profile.matcher.arhitecture	100% (6/6)	61.5% (8/13)	48% (12/25)
com.profile.matcher.assembler	100% (4/4)	41.7% (5/12)	43.4% (33/76)
com.profile.matcher.command	100% (1/1)	100% (2/2)	100% (3/3)
com.profile.matcher.controller	100% (1/1)	100% (2/2)	100% (10/10)
com.profile.matcher.dto.campaign	100% (5/5)	100% (21/21)	100% (21/21)
com.profile.matcher.dto.player	75% (3/4)	77.6% (38/49)	77.6% (38/49)
com.profile.matcher.entity.campaign	100% (1/1)	21.4% (3/14)	21.4% (3/14)
com.profile.matcher.entity.player	100% (5/5)	47.7% (31/65)	47.7% (31/65)
com.profile.matcher.exception	66.7% (2/3)	33.3% (2/6)	33.3% (2/6)
com.profile.matcher.service	100% (3/3)	87.5% (14/16)	76.7% (89/116)
com.profile.matcher.utils	33.3% (1/3)	25% (1/4)	28.6% (2/7)

generated on 2024-01-23 19:53

Figura 5 Raport acoperire cod testare unitară

Pentru analizarea rezultatelor, se poate accesa raportul complet inclus in proiect.

6 CONCLUZII

Arhitectura aplicației web de semnare este una de tip multi-nivel. Avantajul principal în alegerea acestei arhitecturi este faptul că aplicația este ușor de extins pe viitor prin adăugarea de noi module și funcționalități.

Prin folosirea framework-urilor SpringBoot, Spring Data JPA am reușit să implementez o aplicație back-end performantă, prin împărțirea acesteia în niveluri logice, astfel încât fiecare nivel să aibă o singură responsabilitate respectând Principiului Responsabilității Unice, care afirmă că „Un modul ar trebui să fie responsabil față de un singur actor” [1].

7 BIBLIOGRAFIE

[1] R. Martin, Clean architecture: a craftsman's guide to software structure and design, Prentice Hall, 2017.

[2] „Spring Data JPA - Reference Documentation," VMware, Inc. or its affiliates, [Interactiv]. Available: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/. [Accesat 13 01 2023].

[3] „The IoC container," VMware, Inc. or its affiliates, [Interactiv]. Available: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/beans.html. [Accesat January 2023].

8 ANEXE

Api docs:


```
{
    "openapi": "3.0.1",
    "info": {
        "title": "OpenAPI definition",
        "version": "v0"
    },
    "servers": [
        {
            "url": "http://localhost:8080",
            "description": "Generated server url"
        }
    ],
    "paths": {
        "/get_client_config/{player_id}": {
            "get": {
                "tags": [
                    "player-controller"
                ],
                "operationId": "getClientConfig",
                "parameters": [
                    {
                        "name": "player_id",
                        "in": "path",
                        "required": true,
                        "schema": {
                            "type": "string"
```

```
                    }
                }
            ],
            "responses": {
                "200": {
                    "description": "OK",
                    "content": {
                        "*/*": {
                            "schema": {
                                "$ref": "#/components/schemas/PlayerDto"
                            }
                        }
                    }
                }
            }
        }
    },
    "components": {
        "schemas": {
            "ClanDto": {
                "required": [
                    "name"
                ],
                "type": "object",
                "properties": {
                    "id": {
                        "type": "integer",
                        "format": "int64"
                    },
                    "name": {
                        "type": "string"
                    },
                    "startDate": {
                        "type": "string"
                    },
                    "endDate": {
                        "type": "string"
                    }
                }
            },
            "DeviceDto": {
                "type": "object",
                "properties": {
                    "id": {
                        "type": "integer",
                        "format": "int64"
                    },
```

```
                "model": {
                    "type": "string"
                },
                "carrier": {
                    "type": "string"
                },
                "firmware": {
                    "type": "string"
                },
                "startDate": {
                    "type": "string"
                },
                "endDate": {
                    "type": "string"
                }
            }
        },
        "PlayerDto": {
            "required": [
                "credential"
            ],
            "type": "object",
            "properties": {
                "player_id": {
                    "type": "string",
                    "format": "uuid"
                },
                "credential": {
                    "maxLength": 255,
                    "minLength": 0,
                    "type": "string"
                },
                "created": {
                    "type": "string"
                },
                "modified": {
                    "type": "string"
                },
                "last_session": {
                    "type": "string"
                },
                "total_spent": {
                    "type": "number"
                },
                "total_refund": {
                    "type": "number"
                },
                "total_transactions": {
```

```
                "type": "integer",
                "format": "int32"
            },
            "last_purchase": {
                "type": "string"
            },
            "active_campaigns": {
                "type": "array",
                "items": {
                    "type": "string"
                }
            },
            "devices": {
                "type": "array",
                "items": {
                    "$ref": "#/components/schemas/DeviceDto"
                }
            },
            "level": {
                "type": "integer",
                "format": "int32"
            },
            "xp": {
                "type": "integer",
                "format": "int32"
            },
            "total_playtime": {
                "type": "number"
            },
            "country": {
                "type": "string"
            },
            "language": {
                "type": "string"
            },
            "birthdate": {
                "type": "string"
            },
            "gender": {
                "type": "string"
            },
            "inventory": {
                "type": "object",
                "additionalProperties": {
                    "type": "string"
                }
            },
            "clan": {
```

```
                        "$ref": "#/components/schemas/ClanDto"
                    },
                    "get_customField": {
                        "type": "string"
                    }
                }
            }
        }
    }
}
```