Laborator 4 – PPAW – ORM Code First

Raport de Implementare

---

1. ORM - Object Relational Mapping – Code First

Exercițiul 1: Utilizarea conceptului "Code First"

1.1 Proiect Spring Boot Proiectul `password-vault` este de tip Spring Boot Application cu următoarele dependențe:

pom.xml:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
</dependency>
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
</dependency>
```

1.2 Modele definite (Echivalent LibrarieModele) Clasele corespunzătoare entităților sunt definite în `src/main/java/com/ppaw/passwordvault/model/`:

Exemplu: User.java

```
@Entity
@Table(name = "users", schema = "vault_schema", indexes = {
    @Index(name = "idx_users_plan_id", columnList = "service_plan_id"),
    @Index(name = "idx_users_email", columnList = "email")
})
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true, length = 100)
```

```
    private String username;

    @Column(nullable = false, unique = true)
    private String email;

    @Column(nullable = false, name = "password_hash")
    private String passwordHash;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "service_plan_id", nullable = false)
    private ServicePlan servicePlan;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<VaultItem> vaultItems;

    @PrePersist
    protected void onCreate() {
        createdAt = LocalDateTime.now();
        updatedAt = LocalDateTime.now();
    }
}
```

Caracteristici importante: - `@Id` + `@GeneratedValue` pentru chei primare - `@ManyToOne` / `@OneToMany` cu `@JoinColumn` pentru chei străine - `@Column` cu specificații pentru nullable, unique, length - `@PrePersist` pentru automatizare timestamp-uri

1.3 Repository (Echivalent DBContext) Repository-urile JPA extind `JpaRepository`:

UserRepository.java:

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
    Optional<User> findByEmail(String email);

    @Query("SELECT DISTINCT u FROM User u JOIN FETCH u.servicePlan WHERE u.id = :id")
    Optional<User> findByIdWithEagerLoading(@Param("id") Long id);
}
```

1.4 String de Conectare în application.properties application.properties:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/password_vault
spring.datasource.username=postgres
spring.datasource.password=postgres
```

```
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=validate
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.default_schema=vault_schema

spring.flyway.enabled=true
spring.flyway.schemas=vault_schema
spring.flyway.locations=classpath:db/migration
spring.flyway.baseline-on-migrate=true
```

Exercițiul 2: Generarea migrărilor și crearea tabelelor

2.1 Activare migrări (Echivalent Enable-Migrations) Configurare Flyway în application.properties:

```
spring.flyway.enabled=true
spring.flyway.schemas=vault_schema
spring.flyway.locations=classpath:db/migration
spring.flyway.baseline-on-migrate=true
```

Flyway creează automat tabela `flyway_schema_history` pentru evidența migrațiilor.

2.2 Generarea migrărilor (Echivalent Add-Migration) Migrațiile sunt create manual în `src/main/resources/db/migration/`:

V1___20241201_create_schema.sql:

```
-- Flyway Migration V1: Creare Schema
-- Echivalent Enable-Migrations din Entity Framework
CREATE SCHEMA IF NOT EXISTS vault_schema;
```

V2___20241201_create_service_plans.sql:

```
CREATE TABLE IF NOT EXISTS vault_schema.service_plans (
    id BIGSERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE,
    price DECIMAL(10,2) NOT NULL,
    currency VARCHAR(10) NOT NULL DEFAULT 'USD',
    is_active BOOLEAN NOT NULL DEFAULT true,
    created_at TIMESTAMP NOT NULL,
    updated_at TIMESTAMP NOT NULL
);
```

V3___20241201_create_plan_limits.sql:

```sql
CREATE TABLE IF NOT EXISTS vault_schema.plan_limits (
    id BIGSERIAL PRIMARY KEY,
    plan_id BIGINT NOT NULL UNIQUE,
    max_vault_items INTEGER NOT NULL DEFAULT 20,
    -- ... alte coloane
    CONSTRAINT fk_plan_limits_service_plan FOREIGN KEY (plan_id)
        REFERENCES vault_schema.service_plans(id) ON DELETE CASCADE
);
```

Convenții Flyway: - Nume: `V{versiune}__{descriere}.sql` - Versiunea trebuie să fie unică și crescătoare - Underscore dublu (`__`) separă versiunea de descriere

2.3 Aplicarea migrărilor (Echivalent Update-Database) Migrațiile se aplică automat la pornirea aplicației Spring Boot:

```
mvn spring-boot:run
```

Sau verificare manuală:

```sql
SELECT * FROM vault_schema.flyway_schema_history;
```

Exercițiul 3: Seed data în 2 tabele

V9___20241201_seed_service_plans.sql:

```sql
-- Flyway Migration V9: Seed date pentru service_plans și plan_limits
-- Echivalent Configuration.Seed() din Entity Framework
-- IMPORTANT: Datele se inserează în ordine - mai întâi service_plans (părinte), apoi plan_

-- Seed service_plans (tabela părinte - fără FK)
INSERT INTO vault_schema.service_plans (name, price, currency, is_active, created_at, update
VALUES
    ('Free', 0.00, 'USD', true, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP),
    ('Usual', 4.99, 'USD', true, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP),
    ('Premium', 9.99, 'USD', true, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP)
ON CONFLICT (name) DO NOTHING;

-- Seed plan_limits (tabela copil - cu FK către service_plans)
-- Datele se inserează DOAR după ce service_plans sunt inserate
INSERT INTO vault_schema.plan_limits (
    plan_id, max_vault_items, max_password_length, can_export, ...
)
SELECT
    sp.id,
    CASE sp.name
        WHEN 'Free' THEN 20
```

```
        WHEN 'Usual' THEN 200
        WHEN 'Premium' THEN 2000
    END as max_vault_items,
    -- ... alte câmpuri
FROM vault_schema.service_plans sp
WHERE NOT EXISTS (
    SELECT 1 FROM vault_schema.plan_limits pl WHERE pl.plan_id = sp.id
);
```

Răspuns: Care date sunt inserate prima dată? Datele se inserează **în ordinea dependențelor**: 1. **Prima dată**: `service_plans` (tabela părinte, fără FK) 2. **A doua oară**: `plan_limits` (tabela copil, cu FK către `service_plans`)

Acest lucru este necesar pentru a respecta constrângerile de chei străine.

Exercițiul 4: Console Application pentru afișare date

ConsoleDataDisplayRunner.java:

```
@Component
@Order(3)
public class ConsoleDataDisplayRunner implements CommandLineRunner {

    @Autowired
    private ServicePlanRepository servicePlanRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private VaultItemRepository vaultItemRepository;

    @Override
    public void run(String... args) {
        System.out.println("CONSOLE DATA DISPLAY - Afișare date din tabele");

        // Afișează Service Plans
        displayServicePlans();

        // Afișează Users
        displayUsers();

        // Afișează Vault Items
        displayVaultItems();
    }
```

```
    private void displayServicePlans() {
        List<ServicePlan> plans = servicePlanRepository.findAll();
        System.out.printf("%-5s | %-10s | %-10s | %-8s | %-10s%n",
            "ID", "Nume", "Preț", "Monedă", "Activ");

        for (ServicePlan plan : plans) {
            System.out.printf("%-5d | %-10s | %-10s | %-8s | %-10s%n",
                plan.getId(), plan.getName(), plan.getPrice(),
                plan.getCurrency(), plan.getIsActive() ? "DA" : "NU");
        }
    }

    private void displayUsers() {
        List<User> users = userRepository.findAll();
        // ... afișare users
    }
}
```

Exemplu de output:

```
================================================================================
CONSOLE DATA DISPLAY - Afișare date din tabele
================================================================================

  SERVICE PLANS (tabela service_plans):
ID    | Nume       | Preț       | Monedă   | Activ
1     | Free       | 0.00       | USD      | DA
2     | Usual      | 4.99       | USD      | DA
3     | Premium    | 9.99       | USD      | DA
Total: 3 planuri
```

Observații: - Se folosesc Repository-urile JPA pentru acces la date - Rulează automat la pornirea aplicației (`CommandLineRunner`) - Afișează date formatate din multiple tabele

---

2. ORM – Code First – Efectuarea modificărilor

Exercițiul 5: Modificări modele și actualizare baza de date

5.1 Modificări efectuate 1. Adăugare 2 proprietăți la User:

```
// NOUĂ PROPRIETATE 1: Data ultimei autentificări
@Column(name = "last_login_at")
private LocalDateTime lastLoginAt;

// NOUĂ PROPRIETATE 2: Numărul de autentificări
```

```java
@Column(nullable = false, name = "login_count")
private Integer loginCount = 0;
```

2. Modificare tip date în ServicePlan:

```java
// MODIFICARE TIP DATE: de la VARCHAR(3) la VARCHAR(10)
@Column(nullable = false, length = 10)
private String currency = "USD";
```

3. Model nou: AuditLog

```java
@Entity
@Table(name = "audit_logs", schema = "vault_schema")
public class AuditLog {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @Column(nullable = false, length = 50)
    private String action;

    @Column(columnDefinition = "TEXT")
    private String description;

    @Column(length = 45)
    private String ipAddress;

    @Column(nullable = false, updatable = false)
    private LocalDateTime createdAt;
}
```

5.2 Migrație pentru modificări **V10__20241201_modify_entities.sql:**

```sql
-- Flyway Migration V10: Modificări entități
-- Echivalent Add-Migration în Entity Framework

-- 1. Adăugare coloane noi la tabela users
ALTER TABLE vault_schema.users
ADD COLUMN IF NOT EXISTS last_login_at TIMESTAMP,
ADD COLUMN IF NOT EXISTS login_count INTEGER NOT NULL DEFAULT 0;

-- 2. Modificare tip date pentru currency în service_plans
ALTER TABLE vault_schema.service_plans
ALTER COLUMN currency TYPE VARCHAR(10);
```

```sql
-- 3. Creare tabel nou: audit_logs
CREATE TABLE IF NOT EXISTS vault_schema.audit_logs (
    id BIGSERIAL PRIMARY KEY,
    user_id BIGINT NOT NULL,
    action VARCHAR(50) NOT NULL,
    description TEXT,
    ip_address VARCHAR(45),
    created_at TIMESTAMP NOT NULL,
    CONSTRAINT fk_audit_logs_user FOREIGN KEY (user_id)
        REFERENCES vault_schema.users(id) ON DELETE CASCADE
);

-- Indexuri
CREATE INDEX IF NOT EXISTS idx_audit_logs_user_id ON vault_schema.audit_logs(user_id);
CREATE INDEX IF NOT EXISTS idx_audit_logs_action ON vault_schema.audit_logs(action);
```

Analiză migrație: - `ALTER TABLE ... ADD COLUMN` pentru proprietăți noi - `ALTER TABLE ... ALTER COLUMN TYPE` pentru modificare tip - `CREATE TABLE` pentru model nou - `CREATE INDEX` pentru optimizare

5.3 Aplicare migrație Migrația se aplică automat la pornirea aplicației. Verificare:

```sql
-- Verificare coloane noi
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_schema = 'vault_schema'
AND table_name = 'users'
AND column_name IN ('last_login_at', 'login_count');

-- Verificare tabel nou
SELECT * FROM vault_schema.audit_logs;
```

Exercițiul 6: Actualizare cod pentru noile proprietăți

User.java actualizat:

```java
@OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
private List<AuditLog> auditLogs;  // NOUĂ RELAȚIE
```

Repository nou:

```java
@Repository
public interface AuditLogRepository extends JpaRepository<AuditLog, Long> {
    List<AuditLog> findByUserId(Long userId);
    List<AuditLog> findByAction(String action);
```

```
    List<AuditLog> findByCreatedAtBetween(LocalDateTime start, LocalDateTime end);
}
```

Service actualizat:

```
@Service
public class UserService {
    public void recordLogin(Long userId) {
        User user = userRepository.findById(userId).orElseThrow(...);
        user.setLastLoginAt(LocalDateTime.now());
        user.setLoginCount((user.getLoginCount() != null ? user.getLoginCount() : 0) + 1);
        userRepository.save(user);
    }
}
```

---

3. ORM – Lazy loading vs Eager loading

Exercițiul 7: Testare Lazy loading vs Eager loading

LazyEagerLoadingDemo.java:

```
@Component
@Order(4)
public class LazyEagerLoadingDemo implements CommandLineRunner {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private VaultItemRepository vaultItemRepository;

    @Override
    @Transactional
    public void run(String... args) {
        // Demo 1: Lazy Loading (implicit)
        demonstrateLazyLoading(userId);

        // Demo 2: Eager Loading cu JOIN FETCH
        demonstrateEagerLoadingWithJoinFetch(userId);

        // Demo 3: Eager Loading cu @EntityGraph
        demonstrateEagerLoadingWithEntityGraph(userId);
    }

    /**
     * Demo 1: Lazy Loading (implicit)
```

```
     * Relația servicePlan este LAZY - se încarcă doar când e accesată
     */
    @Transactional
    private void demonstrateLazyLoading(Long userId) {
        // Query 1: SELECT * FROM users WHERE id = ?
        User user = userRepository.findById(userId).orElse(null);

        // Query 2: SELECT * FROM service_plans WHERE id = ? (executat LAZY)
        ServicePlan plan = user.getServicePlan(); // Query separat!
    }

    /**
     * Demo 2: Eager Loading cu JOIN FETCH
     * Un singur query cu JOIN
     */
    @Transactional
    private void demonstrateEagerLoadingWithJoinFetch(Long userId) {
        // Un singur query: SELECT u.*, sp.* FROM users u JOIN service_plans sp ON ...
        User user = userRepository.findByIdWithEagerLoading(userId).orElse(null);

        // NU generează query separat - deja încărcat
        ServicePlan plan = user.getServicePlan(); // Deja în memorie!
    }
}
```

UserRepository.java - Metode pentru eager loading:

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {

    // Lazy loading - implicit
    Optional<User> findById(Long id);

    // Eager loading cu JOIN FETCH
    @Query("SELECT DISTINCT u FROM User u JOIN FETCH u.servicePlan LEFT JOIN FETCH u.vaultIt
    Optional<User> findByIdWithEagerLoading(@Param("id") Long id);

    // Eager loading cu @EntityGraph
    @EntityGraph(attributePaths = {"servicePlan", "vaultItems"})
    @Query("SELECT u FROM User u WHERE u.id = :id")
    Optional<User> findByIdWithRelations(@Param("id") Long id);
}
```

User.java - Configurare Lazy:

```
@ManyToOne(fetch = FetchType.LAZY)  // Lazy implicit
@JoinColumn(name = "service_plan_id", nullable = false)
private ServicePlan servicePlan;
```

```java
@OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
private List<VaultItem> vaultItems;  // Lazy implicit pentru @OneToMany
```

Diferențe:

Aspect	Lazy Loading	Eager Loading
Când se încarcă	Doar când e accesat	În același query
Număr query-uri	Multiple (problema N+1)	Unul singur
Configurare	`FetchType.LAZY` (implicit)	`JOIN FETCH` sau `@EntityGraph`
Performanță	Bun pentru relații rare accesate	Bun pentru relații mereu necesare

---

4. Pașii pentru generarea bazei de date

Spring Boot + Flyway

1. Activare migrări:

```
# application.properties
spring.flyway.enabled=true
spring.flyway.schemas=vault_schema
spring.flyway.locations=classpath:db/migration
spring.flyway.baseline-on-migrate=true
spring.jpa.hibernate.ddl-auto=validate
```

2. Creare migrări: - Creează fișiere `V{versiune}__{descriere}.sql` în `src/main/resources/db/migration/` - Exemplu: `V1__20241201_create_schema.sql`

3. Aplicare migrări:

```
mvn spring-boot:run
```

Flyway aplică automat toate migrațiile la pornire.

4. Verificare:

```sql
SELECT * FROM vault_schema.flyway_schema_history;
SELECT table_name FROM information_schema.tables
WHERE table_schema = 'vault_schema';
```

---

5. Setări pentru comutare Lazy/Eager Loading

Opțiunea 1: Modificare în entitate

```java
// LAZY (implicit pentru @ManyToOne)
@ManyToOne(fetch = FetchType.LAZY)
private ServicePlan servicePlan;

// EAGER (nu recomandat - încarcă mereu)
@ManyToOne(fetch = FetchType.EAGER)
private ServicePlan servicePlan;
```

Opțiunea 2: Query cu JOIN FETCH (recomandat)

```java
// Lazy implicit în entity, dar eager doar când ai nevoie
@Query("SELECT u FROM User u JOIN FETCH u.servicePlan WHERE u.id = :id")
Optional<User> findByIdWithPlan(@Param("id") Long id);
```

Opțiunea 3: @EntityGraph (flexibil pentru multiple relații)

```java
@EntityGraph(attributePaths = {"servicePlan", "vaultItems", "auditLogs"})
Optional<User> findById(Long id);
```

Reguli generale: - **Lazy implicit** pe `@OneToMany` și `@ManyToMany` - **Lazy implicit** pe `@ManyToOne` (de obicei) - **Eager doar când e nevoie** - folosește JOIN FETCH sau @EntityGraph - **Evită FetchType.EAGER** în entitate - încarcă mereu, chiar dacă nu ai nevoie

---

6. Backup bazei de date

Backup complet PostgreSQL:

```
pg_dump -h localhost -U postgres -d password_vault -F c -f backup_password_vault_$(date +%Y%
```

Restaurare:

```
pg_restore -h localhost -U postgres -d password_vault -c backup_password_vault_YYYYMMDD_HHMM
```

Backup SQL plain:

```
pg_dump -h localhost -U postgres -d password_vault -f backup.sql
```

---

7. Testare API REST

Exercițiul 8: Testare endpoint-uri API

Pentru a demonstra funcționalitatea completă a aplicației, au fost testate toate endpoint-urile API REST create.

7.1 Verificare Stare Aplicație Request:

```
curl http://localhost:8080/api/health
```

Răspuns:


```
{
  "success": true,
  "message": "Application is healthy",
  "data": {
    "application": "Password Vault API",
    "status": "UP",
    "timestamp": "2026-01-12T13:17:07.194486989"
  },
  "timestamp": "2026-01-12T13:17:07.194512653"
}
```


7.2 Statistici Request:

```
curl http://localhost:8080/api/stats
```

Răspuns:


```
{
  "success": true,
  "message": "Statistics retrieved successfully",
  "data": {
    "totalVaultItems": 1,
    "totalUsers": 2,
    "totalServicePlans": 3,
    "totalAuditLogs": 5,
    "activeUsers": 2
  },
  "timestamp": "2026-01-12T13:17:08.182844709"
}
```


7.3 Planuri de Servicii Request:

```
curl http://localhost:8080/api/service-plans
```

Răspuns:


```
{
  "success": true,
  "message": "Service plans retrieved successfully",
  "data": [
    {
      "id": 1,
      "name": "Free",
      "price": 0.00,
```

```
      "currency": "USD",
      "isActive": true,
      "createdAt": "2026-01-12T08:46:44.144392",
      "updatedAt": "2026-01-12T08:46:44.144422",
      "limits": {
        "id": 1,
        "planId": 1,
        "maxVaultItems": 20,
        "maxPasswordLength": 16,
        "canExport": false,
        "canImport": false,
        "canShare": false,
        "maxHistoryVersions": 0,
        "canAttachments": false,
        "maxDevices": 1,
        "excludeAmbiguous": false
      }
    },
    {
      "id": 2,
      "name": "Usual",
      "price": 4.99,
      "currency": "USD",
      "isActive": true,
      "limits": {
        "maxVaultItems": 200,
        "maxPasswordLength": 32,
        "canExport": true
      }
    },
    {
      "id": 3,
      "name": "Premium",
      "price": 9.99,
      "currency": "USD",
      "isActive": true,
      "limits": {
        "maxVaultItems": 2000,
        "maxPasswordLength": 64,
        "canExport": true,
        "canImport": true,
        "canShare": true
      }
    }
  ],
  "timestamp": "2026-01-12T13:17:09.202374249"
```

```
}
```


Obținere Plan după ID:

```
curl http://localhost:8080/api/service-plans/1
```

Răspuns:


```
{
  "success": true,
  "message": "Service plan retrieved successfully",
  "data": {
    "id": 1,
    "name": "Free",
    "price": 0.00,
    "currency": "USD",
    "isActive": true,
    "limits": {
      "id": 1,
      "planId": 1,
      "maxVaultItems": 20,
      "maxPasswordLength": 16,
      "canExport": false
    }
  },
  "timestamp": "2026-01-12T13:17:10.080384654"
}
```


7.4 Utilizatori Obținere Toți Utilizatorii:

```
curl http://localhost:8080/api/users
```

Răspuns:


```
{
  "success": true,
  "message": "Users retrieved successfully",
  "data": [
    {
      "id": 1,
      "username": "vovan",
      "email": "vovan@vovan.vovan",
      "servicePlanId": 1,
      "servicePlanName": "Free",
      "isActive": true,
      "lastLoginAt": "2026-01-12T12:58:51.049071",
      "loginCount": 1,
      "createdAt": "2026-01-12T12:54:39.402655",
      "updatedAt": "2026-01-12T12:58:51.078131"
```

```
    },
    {
      "id": 2,
      "username": "test_user_api",
      "email": "test_api@example.com",
      "servicePlanId": 1,
      "servicePlanName": "Free",
      "isActive": true,
      "lastLoginAt": null,
      "loginCount": 0,
      "createdAt": "2026-01-12T13:17:00.819171",
      "updatedAt": "2026-01-12T13:17:00.819186"
    }
  ],
  "timestamp": "2026-01-12T13:17:11.074149936"
}
```


Obținere Utilizator după ID:

```
curl http://localhost:8080/api/users/2
```

Răspuns:


```
{
  "success": true,
  "message": "User retrieved successfully",
  "data": {
    "id": 2,
    "username": "test_user_api",
    "email": "test_api@example.com",
    "servicePlanId": 1,
    "servicePlanName": "Free",
    "isActive": true,
    "lastLoginAt": "2026-01-12T13:17:22.736652",
    "loginCount": 1,
    "createdAt": "2026-01-12T13:17:00.819171",
    "updatedAt": "2026-01-12T13:17:22.744222"
  },
  "timestamp": "2026-01-12T13:17:23.711880839"
}
```


Creare Utilizator:

```
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{
    "username": "test_user_api",
    "email": "test_api@example.com",
    "password": "password123",
```

```
    "servicePlanId": 1
  }'
```

Răspuns (username duplicat):

```
{
  "success": false,
  "message": "Username already exists",
  "data": null,
  "timestamp": "2026-01-12T13:17:12.013212101"
}
```

Actualizare Utilizator:

```
curl -X PUT http://localhost:8080/api/users/2 \
  -H "Content-Type: application/json" \
  -d '{"username": "updated_user", "isActive": true}'
```

Răspuns:

```
{
  "success": true,
  "message": "User updated successfully",
  "data": {
    "id": 2,
    "username": "updated_user",
    "email": "test_api@example.com",
    "servicePlanId": 1,
    "servicePlanName": "Free",
    "isActive": true,
    "lastLoginAt": "2026-01-12T13:17:22.736652",
    "loginCount": 1
  },
  "timestamp": "2026-01-12T13:17:28.018703524"
}
```

7.5 Autentificare (Login) Login cu Succes:

```
curl -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "test_user_api",
    "password": "password123"
  }'
```

Răspuns:

```
{
  "success": true,
  "message": "Login successful",
```

```
  "data": {
    "userId": 2,
    "username": "test_user_api",
    "email": "test_api@example.com",
    "servicePlanId": 1,
    "servicePlanName": "Free",
    "lastLoginAt": "2026-01-12T13:17:22.73665248",
    "loginCount": 1,
    "success": true
  },
  "timestamp": "2026-01-12T13:17:22.749619125"
}
```

Login Eșuat (Credențiale Invalide):

```
curl -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "username": "wrong_user",
    "password": "wrong"
  }'
```

Răspuns:

```
{
  "success": false,
  "message": "Invalid username or password",
  "data": null,
  "timestamp": "2026-01-12T13:17:29.121377755"
}
```

7.6 Item-uri din Vault Creare Item în Vault:

```
curl -X POST http://localhost:8080/api/users/2/vault-items \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Gmail Account",
    "username": "test@gmail.com",
    "password": "myPassword123",
    "url": "https://gmail.com",
    "folder": "Email",
    "isFavorite": true
  }'
```

Răspuns:

```
{
  "success": true,
  "message": "Vault item created successfully",
```

```
  "data": {
    "id": 2,
    "userId": 2,
    "title": "Gmail Account",
    "username": "test@gmail.com",
    "url": "https://gmail.com",
    "notes": null,
    "folder": "Email",
    "tags": null,
    "isFavorite": true,
    "createdAt": "2026-01-12T13:17:24.832160336",
    "updatedAt": "2026-01-12T13:17:24.832176874"
  },
  "timestamp": "2026-01-12T13:17:24.842922631"
}
```


Obținere Toate Item-urile din Vault:

```
curl http://localhost:8080/api/users/2/vault-items
```

Răspuns:


```
{
  "success": true,
  "message": "Vault items retrieved successfully",
  "data": [
    {
      "id": 2,
      "userId": 2,
      "title": "Gmail Account",
      "username": "test@gmail.com",
      "url": "https://gmail.com",
      "notes": null,
      "folder": "Email",
      "tags": null,
      "isFavorite": true,
      "createdAt": "2026-01-12T13:17:24.83216",
      "updatedAt": "2026-01-12T13:17:24.832177"
    }
  ],
  "timestamp": "2026-01-12T13:17:25.864813434"
}
```


7.7 Log-uri de Audit Obținere Log-uri de Audit după Utilizator:

```
curl http://localhost:8080/api/audit-logs/user/2
```

Răspuns:

```
{
  "success": true,
  "message": "Audit logs retrieved successfully",
  "data": [
    {
      "id": 5,
      "userId": 2,
      "username": "test_user_api",
      "action": "USER_CREATED",
      "description": "User account created",
      "ipAddress": null,
      "createdAt": "2026-01-12T13:17:00.824766"
    },
    {
      "id": 6,
      "userId": 2,
      "username": "test_user_api",
      "action": "LOGIN",
      "description": "User logged in",
      "ipAddress": null,
      "createdAt": "2026-01-12T13:17:22.737296"
    },
    {
      "id": 7,
      "userId": 2,
      "username": "test_user_api",
      "action": "CREATE_VAULT_ITEM",
      "description": "Created vault item: Gmail Account",
      "ipAddress": null,
      "createdAt": "2026-01-12T13:17:24.834955"
    }
  ],
  "timestamp": "2026-01-12T13:17:26.925343388"
}
```


7.8 Rezultate Testare **Verificare Stare Aplicație**: Funcționează corect, returnează status UP

Statistici: Returnează statistici corecte din baza de date

Planuri de Servicii: Listează toate planurile cu limitările asociate

Utilizatori CRUD: Creare, Citire, Actualizare funcționează corect

Autentificare: Login verifică credențialele și actualizează loginCount

Item-uri din Vault: Creare și Citire funcționează corect

Log-uri de Audit: Toate acțiunile sunt înregistrate corect

Gestionare Erori: Validări și mesaje de eroare funcționează corect

Observații: - Toate răspunsurile folosesc formatul standard `ApiResponse<T>`

- Validările funcționează (username duplicat, credențiale invalide) - Logging de audit este automat pentru toate acțiunile importante - Login actualizează corect `lastLoginAt` și `loginCount`

Tehnologii utilizate: - Spring Boot 3.2.0 - JPA/Hibernate - Flyway pentru migrații - PostgreSQL - Spring Data JPA - REST Controllers - Bean Validation