

# **PROIECT APLICATII WEB JAVA**

## **(AWJ)**

Aplicatie pentru Gestionarea Bugetului Studentesc  
**(StudentBudget)**

**Student:** Petre Adrian

**Grupa:** 331AB

Ianuarie 2026

# Cuprins

<b>1 Specificarea Funcționalităților</b>	<b>3</b>
1.1 Descriere Generală . . . . .	3
1.2 Funcționalități de Bază . . . . .	3
1.3 Funcționalități Suplimentare (Procesare Java) . . . . .	3
<b>2 Arhitectura și Descrierea Claselor</b>	<b>4</b>
2.1 Structura MVC . . . . .	4
2.2 Descrierea Claselor Principale . . . . .	4
2.2.1 Tranzactie.java (Model) . . . . .	4
2.2.2 MainController.java (Controller) . . . . .	4
2.2.3 TranzactieService.java (Service) . . . . .	4
<b>3 Interfața Grafică (View)</b>	<b>5</b>
3.1 Dashboard și Statistici . . . . .	5
3.2 Gestirea Tranzacțiilor (CRUD și Filtrare) . . . . .	5
3.3 Validarea Datelor . . . . .	6
3.4 Confirmarea Acțiunilor . . . . .	6
<b>4 Testare și Depanare</b>	<b>7</b>
4.1 Testare Funcțională (Black-box) . . . . .	7
4.2 Testare API REST (Postman) . . . . .	7
4.3 Idei de îmbunătățire . . . . .	7

# 1 Specificarea Funcționalităților

## 1.1 Descriere Generală

Aplicația **StudentBudget** este o soluție web dezvoltată pe platforma Java Enterprise (Spring Boot), destinată studenților pentru gestionarea eficientă a finanțelor personale. Arhitectura respectă şablonul **MVC (Model-View-Controller)**, separând logica de business de interfața cu utilizatorul.

## 1.2 Funcționalități de Bază

Conform obiectivelor proiectului, aplicația include:

- **Gestiunea Colecției de Obiecte:** Utilizatorul poate gestiona entități de tip **Tranzactie și Cont**.
- **Operații CRUD Complete:**
  - *Create*: Adăugarea de noi venituri/cheltuieli și conturi.
  - *Read*: Vizualizarea listei de tranzacții și a balanței curente.
  - *Update*: Modificarea descrierilor sau sumelor existente.
  - *Delete*: Stergerea elementelor din colecție cu mecanism de siguranță.
- **Interfață Web Dinamică:** Randare server-side folosind motorul de template-uri **Thymeleaf**.

## 1.3 Funcționalități Suplimentare (Procesare Java)

De asemenea, au fost implementate următoarele funcționalități avansate de procesare și logică în Java:

1. **Filtrare Avansată:** Posibilitatea de a filtra lista de tranzacții afișate în funcție de o sumă minimă introdusă de utilizator. Logica de filtrare este gestionată în Backend (Stream API).
2. **Calcul Agregat (Procesare):** Calculul automat al balanței totale a studentului prin iterarea și însumarea soldurilor tuturor conturilor asociate.
3. **REST API pentru Integrări:** Expunerea datelor în format JSON pentru consum extern și testare automatizată.
4. **Validare Strictă (Backend & Frontend):** Implementarea unor reguli de validare pentru a preveni erorile de input (ex: interzicerea caracterelor non-numerice în câmpurile de sumă).
5. **Design Modern (Glassmorphism):** Implementarea unei interfețe grafice responsive și moderne folosind CSS avansat.

## 2 Arhitectura și Descrierea Claselor

### 2.1 Structura MVC

Aplicația este structurată în pachete conform standardelor Spring [1]:

- `com.bugetstudent.model` - Entitățile JPA (Model).
- `com.bugetstudent.repository` - Interfețele pentru persistența datelor.
- `com.bugetstudent.service` - Logica de business.
- `com.bugetstudent.controller` - Gestionarea cererilor HTTP.

### 2.2 Descrierea Claselor Principale

#### 2.2.1 Tranzactie.java (Model)

Clasa care definește obiectul principal din colecție. Folosește adnotări JPA (`@Entity`, `@ManyToOne`) pentru a stabili relațiile cu Conturile și Categoriile.

#### 2.2.2 MainController.java (Controller)

Gestionează rutele principale (`/home`, `/tranzactii`).

- **Metoda showHomePage:** Pregătește datele pentru Dashboard. Apelează service-ul pentru a calcula balanța totală și trimită datele către Thymeleaf prin obiectul Model.
- **Metoda addTranzactie:** Primește datele din formular, le validează și apelează Service-ul pentru salvare.

#### 2.2.3 TranzactieService.java (Service)

Conține logica de "procesare" tranzacțională.

- **Metoda saveTranzactie:** Nu doar salvează tranzacția, ci actualizează automat și balanța contului asociat (scade sau adună suma), asigurând consistența datelor.

```

20  public class MainController {
21
22      private ContService contService;
23
24      // ===== ZONA DASHBOARD (ACASA) =====
25      @GetMapping("/home")
26      public String showHomePage(HttpServletRequest session, Model model) {
27          Integer userId = (Integer) session.getAttribute("user_id");
28          if (userId == null) return "redirect:/";
29
30          // 1. Balanța Totală
31          List<Cont> conturi = contService.getConturiByStudent(userId);
32          double totalBalanta = 0.0;
33          for (Cont c : conturi) {
34              totalBalanta += c.getBalanta();
35          }
36          model.addAttribute("totalBalanta", totalBalanta);
37
38          // 2. STATISTICI NOI
39          // Lista de obiecte [NumereCategorii, SumaTotala]
40          List<Object[]> statistici = tranzactieService.getStatisticiCategorii(userId);
41          model.addAttribute("statistici", statistici);
42
43          return "home";
44      }
45
46      // ===== ZONA TRANZACȚII =====
47
48      @GetMapping("/tranzactii")
49      public String showTranzactii(@RequestParam(required = false) Double minSuma,
50          HttpServletRequest session,
51          Model model) {
52          ...
53      }
54
55  }

```

Figura 1: Exemplu de cod din Controller (Spring MVC)

### 3 Interfața Grafică (View)

Interfața este realizată cu HTML5 și Thymeleaf [2], stilizată cu CSS personalizat.

#### 3.1 Dashboard și Statistici

Pagina principală afișează datele procesate în backend: balanța totală și un rezumat al cheltuielilor.

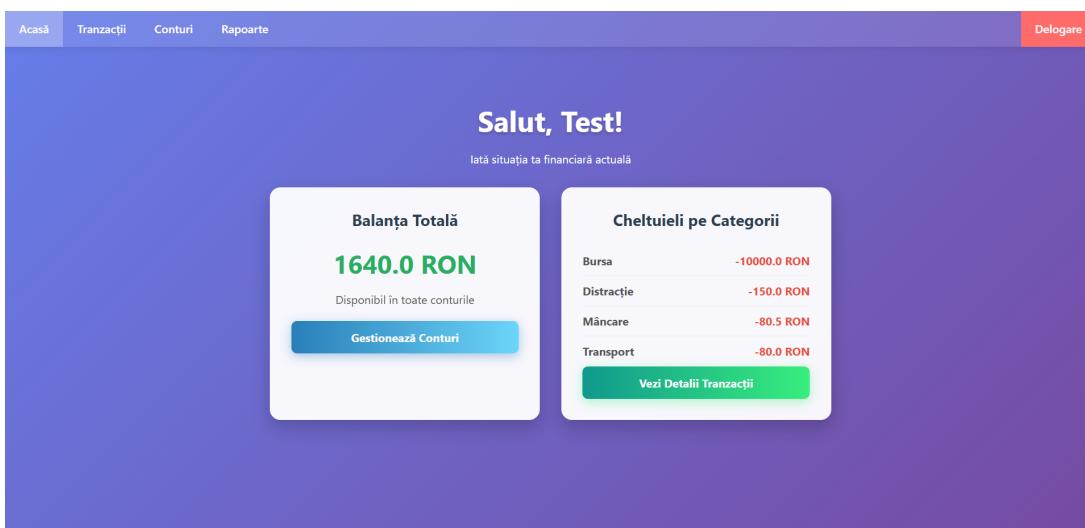


Figura 2: Dashboard-ul aplicației (Date calculate dinamic)

#### 3.2 Gestiunea Tranzacțiilor (CRUD și Filtrare)

Utilizatorul are control total asupra colecției de date. Mai jos sunt prezentate funcționalitățile de Vizualizare, Filtrare și Modificare.

The screenshot shows a transaction list with the following columns: Data, Descriere, Categ., Cont, Suma, and Acțiuni. The transactions listed are:

Data	Descriere	Categ.	Cont	Suma	Acțiuni
09-01 18:46	cafea	Mâncare	Cash Portofel	-10.0 RON	Desc. nouă Save X
05-11 08:15	Abonament autobuz	Transport	Card BCR	-80.0 RON	Desc. nouă Save X
04-11 18:00	Covrigi	Mâncare	Cash Portofel	-25.0 RON	Desc. nouă Save X
03-11 12:30	Prânz cantină	Mâncare	Card BCR	-45.5 RON	Desc. nouă Save X
01-11 10:00	Bani de la părinți	Părinți	Card BCR	700.0 RON	Desc. nouă Save X

Figura 3: Lista de tranzacții cu funcționalitate de Filtrare activă

The screenshot shows a single transaction row with the following details:

Data	Descriere	Categ.	Cont	Suma	Acțiuni
09-01 18:46	cafea	Mâncare	Cash Portofel	-10.0 RON	cafea și suc Save X

Figura 4: Formularul de Modificare (Update) a unui element existent

### 3.3 Validarea Datelor

Aplicația oferă feedback vizual imediat în cazul introducerii unor date eronate, respectând cerința de validare a câmpurilor.

The screenshot shows an error message for an input field. The message reads: "Trebuie să folosești o descriere validă".

Figura 5: Mesaj de eroare la validare (Input invalid)

### 3.4 Confirmarea Acțiunilor

Pentru a preveni ștergerile accidentale este implementat un mecanism de confirmare JavaScript integrat cu backend-ul Java.

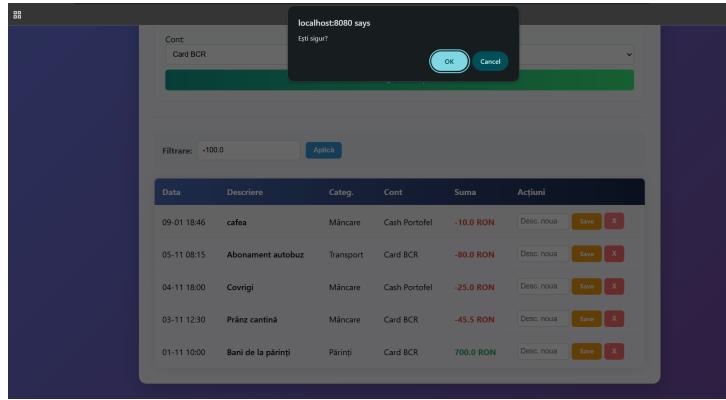


Figura 6: Protectie la stergerea unui obiect din colectie

4 Testare și Depanare

#### 4.1 Testare Funcțională (Black-box)

S-au verificat manual toate scenariile de utilizare (Happy Flow și Error Cases):

- **Adăugare:** Verificarea actualizării automate a balanței în Dashboard după o tranzacție nouă.
  - **Validare:** Introducerea de caractere text în câmpul "Sumă" blochează salvarea, conform așteptărilor.
  - **Filtrare:** Lista afișează corect doar elementele care respectă criteriile de filtrare.

## 4.2 Testare API REST (Postman)

Pentru a verifica integritatea datelor și expunerea corectă a endpoint-urilor, s-a utilizat utilitarul **Postman**, conform recomandărilor din fișa proiectului. S-a testat ruta GET /api/tranzactii, verificând structura JSON returnată și codul de răspuns HTTP 200 OK.

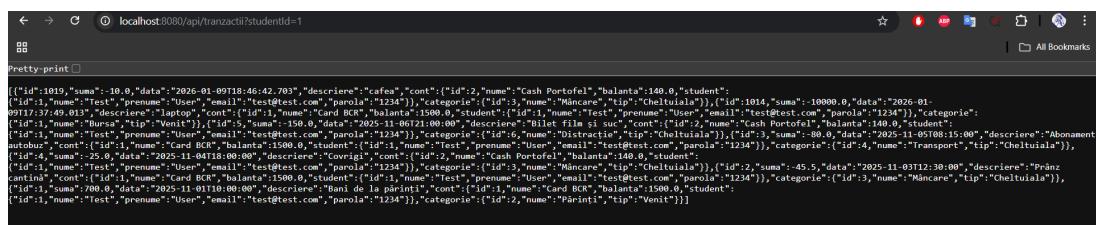


Figura 7: Depanarea si verificarea API-ului REST folosind Postman

### **4.3 Idei de îmbunătățire**

Pentru dezvoltarea ulterioară, propun următoarele 5 funcționalități:

1. **Grafice Vizuale:** Implementarea bibliotecii Chart.js pentru vizualizarea procentuală a cheltuielilor.

2. **Export Date:** Generarea unui raport PDF cu istoricul tranzacțiilor direct din Java (folosind iText).
3. **Bugete Lunare:** Posibilitatea de a seta o limită de cheltuieli pe lună și alertarea utilizatorului.
4. **Autentificare JWT:** Trecerea la o autentificare stateless pentru securitate sporită a API-ului.
5. **Mod Dark/Light:** Implementarea schimbării dinamice a temei vizuale.

## Bibliografie

- [1] *Spring Boot Official Documentation* <https://docs.spring.io/spring-boot/index.html>
- [2] *Thymeleaf Documentation* <https://www.thymeleaf.org/documentation.html>
- [3] *Baeldung - Spring Boot & JPA Tutorials* <https://www.baeldung.com/>
- [4] *Microsoft SQL Server Documentation* <https://learn.microsoft.com/en-us/sql/>