KARTA PROJEKTU I SPECYFIKACJA WYMAGAŃ

Budget Buddy – Aplikacja do zarządzania budżetem osobistym

Akronim: BudgetBuddy / BB

Wersja dokumentu: 2.0 (wersja rozszerzona)
Data utworzenia: 19.10.2025 (prototyp) / 14.02.2026 (aktualizacja)
Jednostka dydaktyczna: WSPA
Zleceniodawca / Sponsor: mgr Wojciech Moniuszko (prowadzący)
Charakter dokumentu: Karta projektu + SRS (wymagania funkcjonalne i niefunkcjonalne)
Repozytorium: GitHub (wspólne repo projektu)
Autorzy (zespół projektowy): Justyna Turowska, Damian Wąsiewicz, Oskar Wójcicki, Aniela Wróbel.

Spis treści

1 Streszczenie projektu

Budget Buddy to aplikacja webowa wspierająca użytkowników w planowaniu i kontroli finansów osobistych. System pozwala rejestrować transakcje (przychody i wydatki), przypisywać je do kategorii, planować budżety miesięczne, analizować strukturę wydatków oraz generować raporty i wykresy. Dokument łączy kartę projektu oraz specyfikację wymagań (SRS) zgodnie z wytycznymi modułu Analiza wymagań.

2 Uzasadnienie biznesowe i edukacyjne

2.1 Problem

Wiele osób ma trudność z bieżącą kontrolą wydatków i planowaniem budżetu. Typowe problemy to brak przejrzystości w historii transakcji, brak informacji o strukturze kosztów oraz brak narzędzi ostrzegających o przekroczeniu założonych limitów.

2.2 Cel biznesowy

Celem biznesowym jest dostarczenie prostego w obsłudze narzędzia do ewidencji i analizy finansów osobistych, wspierającego decyzje o oszczędzaniu i planowaniu wydatków.

2.3 Cel edukacyjny

Celem edukacyjnym projektu jest praktyczne zastosowanie procesu analizy wymagań, identyfikacji interesariuszy, modelowania UML, projektowania bazy danych oraz wytwarzania oprogramowania w zespole w podziale na role.

3 Cele projektu i mierniki sukcesu

3.1 Cel główny

Stworzenie intuicyjnej aplikacji webowej umożliwiającej użytkownikom efektywne planowanie, monitorowanie i analizowanie wydatków oraz przychodów w celu lepszego zarządzania finansami osobistymi.

3.2 Cele szczegółowe

- Notowanie transakcji (przychody/wydatki) z podziałem na kategorie.
- Analiza wydatków – raporty i wykresy w wybranym okresie.
- Planowanie budżetu – tworzenie planów i śledzenie realizacji.
- Powiadomienia – informowanie o zbliżających się płatnościach lub przekroczeniu limitu.

- Eksport danych – CSV/PDF/Excel (na potrzeby raportowania i archiwizacji).

3.3 Kryteria sukcesu

- Wszystkie wymagania funkcjonalne oznaczone jako Must zostały zaimplementowane.
- Aplikacja uruchamia się lokalnie i działa poprawnie (brak błędów krytycznych).
- Dokumentacja zawiera kompletny zestaw diagramów UML oraz ERD.
- Testy akceptacyjne zakończone wynikiem pozytywnym.
- Projekt zrealizowany w wyznaczonym harmonogramie i pozytywnie oceniony przez prowadzącego.

4 Zakres projektu

4.1 Zakres w projekcie (In-Scope)

- Opracowanie wymagań funkcjonalnych i niefunkcjonalnych.
- Zaprojektowanie systemu z wykorzystaniem języka Python oraz modelowania UML.
- Implementacja aplikacji webowej.
- Utworzenie relacyjnej bazy danych SQL.
- Stworzenie dokumentacji projektowej i technicznej.
- Przeprowadzenie testów i prezentacji prototypu.

4.2 Poza zakresem (Out-of-Scope)

- Połączenie projektu z inną grupą i stworzenie aplikacji mobilnej.
- Integracje z bankami (API bankowe) oraz automatyczny import historii transakcji.
- Zaawansowana księgowość (np. VAT, faktury) – poza celem aplikacji osobistej.

4.3 Założenia i ograniczenia

- System jest prototypem edukacyjnym, uruchamianym lokalnie lub w środowisku testowym.
- Dane finansowe są danymi wrażliwymi – wymagane jest zapewnienie izolacji danych użytkowników i bezpiecznego logowania.
- Brak integracji z zewnętrznymi źródłami danych oznacza, że użytkownik wprowadza transakcje ręcznie.

5 Interesariusze i potrzeby

Zgodnie z wytycznymi modułu Analiza wymagań, interesariusze zostali zidentyfikowani i przypisano im potrzeby oraz priorytety (Must/Should/Could).

Interesariusz	Rola	Potrzeby / oczekiwania	Priorytet
Użytkownik końcowy	Korzysta z aplikacji	Prosty interfejs, szybkie dodawanie transakcji, raporty	Must
Administrator (rola systemowa)	Zarządza dostępem	Bezpieczeństwo, możliwość usunięcia konta, utrzymanie danych	Must
Zleceniodawca / Prowadzący	Sponsor / ocenia	Kompletność dokumentacji, zgodność z wytycznymi, działający prototyp	Must
Zespół projektowy	Wytwarza system	Jasny podział ról, plan pracy, narzędzia współpracy	Should
Użytkownicy zewnętrzni (testerzy)	Ocena użyteczności	Czytelność wykresów, zrozumiałe raporty	Could

6 Zespół projektowy, role i odpowiedzialności

6.1 Skład zespołu

Imię i nazwisko	Rola	Zakres odpowiedzialności
Justyna Turowska	Project Manager	Harmonogram, koordynacja, raportowanie, zarządzanie ryzykiem
Justyna Turowska	Analityk systemowy	Wymagania, UML, Use Case, walidacja wymagań
Damian Wąsiewicz	Backend Developer	Logika biznesowa Django, autoryzacja, integracja DB
Oskar Wójcicki	Frontend Developer	Interfejs HTML/CSS/Bootstrap, widoki raportów
Oskar Wójcicki	Security Engineer	Bezpieczeństwo danych, polityki dostępu
Aniela Wróbel	DBA	Projekt PostgreSQL, migracje, integralność danych
Damian Wąsiewicz	QA Engineer	Testy funkcjonalne i akceptacyjne

Aniela Wróbel	Dokumentalista	Redakcja dokumentacji, instrukcje uruchomienia

7 Zasoby i narzędzia

Kategoria	Narzędzie	Uzasadnienie
Repozytorium	GitHub	Kontrola wersji, code review
Programowanie	Python (Django)	Szybkie prototypowanie, ORM, bezpieczeństwo
Frontend	HTML/CSS/Bootstrap	Responsywność i gotowe komponenty UI
Baza danych	PostgreSQL	Relacyjność, ACID, stabilność
Komunikacja	Discord	Szybka komunikacja zespołu

8 Specyfikacja wymagań

8.1 Priorytety (MoSCoW)

Must – konieczne, Should – istotne, Could – opcjonalne, Won't – poza zakresem.

8.2 Wymagania funkcjonalne

ID	Nazwa	Opis	Priorytet
F1	Rejestracja	Utworzenie konta z unikalnym e-mailem	Must
F2	Logowanie	Autoryzacja i sesja użytkownika	Must
F3	Dodanie transakcji	Dodanie przychodu/ wydatku z kategorią	Must
F4	Edycja/usunięcie	Modyfikacja własnych transakcji	Must
F5	Budżet	Ustawienie limitu miesięcznego	Must
F6	Raporty	Generowanie raportów okresowych	Should

8.3 Wymagania niefunkcjonalne

ID	Kategoria	Wymaganie	Weryfikacja
NF1	Wydajność	Czas odpowiedzi dla dodania transakcji ≤ 2 s (P95) w środowisku testowym	Test wydajności

NF2	Dostępność	Dostępność środowiska demo na poziomie 99% w okresie prezentacji	Monitoring/logi
NF3	Bezpieczeństwo	Hasła przechowywane jako hash (rekomendacja: bcrypt)	Przegląd kodu
NF4	Bezpieczeństwo	Izolacja danych – dostęp wyłącznie do własnych rekordów	Test autoryzacji
NF5	Użyteczność	Interfejs responsywny (1366×768 oraz 360×640)	Testy UI
NF6	Integralność	Operacje zapisu transakcji są transakcyjne (brak częściowego zapisu)	Testy DB

9 Przypadki użycia (Use Case)

9.1 Rejestracja użytkownika

Aktor: Użytkownik niezalogowany

Scenariusz główny:

1. Użytkownik podaje e-mail i hasło.
2. System waliduje dane.
3. System zapisuje konto w bazie.
4. Użytkownik otrzymuje potwierdzenie.

Scenariusz alternatywny:

- E-mail już istnieje – komunikat o błędzie.
- Hasło niespełniające wymagań – komunikat walidacyjny.

9.2 Dodanie transakcji

Aktor: Użytkownik zalogowany

Scenariusz główny:

1. Użytkownik wybiera „Dodaj transakcję”.
2. Wprowadza kwotę, datę, typ i kategorię.
3. System waliduje dane.
4. System zapisuje rekord w bazie.
5. Lista transakcji zostaje zaktualizowana.

9.3 Ustawienie limitu budżetu

1. Użytkownik przechodzi do sekcji „Budżet”.
2. Wybiera okres i kategorię.
3. Wprowadza limit kwotowy.
4. System zapisuje limit i monitoruje przekroczenia.

10 Architektura systemu

Aplikacja posiada architekturę warstwową:

- Warstwa prezentacji (HTML/CSS/Bootstrap)
- Warstwa logiki biznesowej (Django)
- Warstwa dostępu do danych (ORM)
- Baza danych PostgreSQL

Separacja warstw umożliwia łatwe testowanie oraz rozwój systemu.

11 Strategia testów

11.1 Rodzaje testów

- Testy jednostkowe
- Testy integracyjne
- Testy funkcjonalne
- Testy akceptacyjne
- Testy bezpieczeństwa

11.2 Przykładowe testy akceptacyjne

ID	Wymaganie	Opis testu	Kryterium
TA1	F2	Logowanie poprawnymi danymi	Dostęp do panelu
TA2	F3	Dodanie wydatku 50 zł	Transakcja widoczna na liście
TA3	F5	Ustawienie limitu 500 zł	Limit zapisany poprawnie

TA4	NF4	Próba dostępu do cudzych danych	Brak dostępu (403)

12 Harmonogram

Etap	Zakres	Czas	Rezultat
1	Tworzenie zespołu i karta projektu	Tydzień 1	Karta projektu
2	Analiza wymagań	Tydzień 2	Dokument wymagań
3	Projekt UML	Tydzień 3–4	Diagramy UML
4	Implementacja prototypu	Tydzień 6–7	Aplikacja v1
5	Testy i poprawki	Tydzień 8	Raport testów
6	Dokumentacja końcowa	Tydzień 9–10	Prezentacja

13 Analiza ryzyka

Nr	Ryzyko	Prawdop.	Skutek	Działania
1	Opóźnienia	Średnie	Wysoki	Cotygodniowe spotkania
2	Brak doświadczenia	Wysokie	Średni	Code review, podział zadań
3	Utrata danych	Niskie	Wysoki	Backup repozytorium
4	Błędy finansowe	Średnie	Wysoki	Testy jednostkowe

14 Rezultaty projektu

- Prototyp aplikacji BudgetBuddy
- Dokumentacja SRS + UML + ERD
- Instrukcja uruchomienia
- Raport testów
- Prezentacja końcowa

15 Akceptacja projektu

Funkcja	Imię i nazwisko	Data	Podpis
Kierownik projektu	Justyna Turowska	14.02.2026	
Prowadzący	mgr Wojciech Moniuszko	14.02.2026	

16 Uwagi końcowe

- Dokument przechowywany w repozytorium projektu.
- Aktualizacja wersji wymaga zgody kierownika projektu.
- Każdy członek zespołu ma obowiązek zapoznać się z dokumentem.