

SOFTWARE DOCUMENTATION · V1.0

CampusCoin

A vivid personal-finance ledger built for students: allowance in, canteen out, and every naira accounted for.

Django REST + React single-page application with MySQL persistence, JWT authentication, budgeting, savings goals, bill splitting, spending analytics and an intent-driven money assistant.

Project: Campus Coin — Student Money Manager

Prepared: 27 September 2026

Audience: Assessors, instructors and future maintainers

Companion artefact: CampusCoin_Walkthrough.mp4 (narrated feature tour)

Contents

1. Introduction and Purpose
2. System Overview and Architecture
3. Technology Stack
4. Installation and Setup
5. Database Schema
6. API Reference
7. Feature Guide — Page by Page
8. The Money Roommate Assistant
9. User Interface, Accessibility and Localisation
10. Security and Authentication
11. Testing and Verification
12. Troubleshooting and Maintenance

1. Introduction and Purpose

Campus Coin is a full-stack web application that helps students manage personal money without needing a bank connection. Users record income (allowance, scholarships, side gigs) and expenses (canteen, hostel, transport, data), set per-category budgets, save towards named goals, split shared bills with roommates, and review their habits through charts, a spending heatmap and a what-if simulator.

The purpose of this document is to describe every functional aspect of the system: its architecture, data model, application programming interface (API), user-facing features, security model and maintenance procedures. It is intended to be read alongside the narrated walkthrough video shipped in the same documentation folder.

1.1 Design goals

- **Honest and manual:** the user enters every figure; nothing is scraped from a bank.
- **Immediate feedback:** dashboards, meters and alerts react the moment data changes.
- **Plain language:** analytics are expressed in sentences a student can act on, not raw statistics.
- **Private by default:** every record is owned by exactly one user and never visible to others.
- **Responsive and accessible:** usable from a phone in a lecture hall or a desktop in the hostel.

2. System Overview and Architecture

The system follows a classic three-tier client-server architecture:

- **Presentation tier:** a React single-page application (SPA) served by the Vite development server. It owns all routing, rendering and user interaction, and talks to the API over HTTP/JSON.
- **Application tier:** a Django project exposing a REST API through Django REST Framework (DRF). It authenticates requests with JSON Web Tokens (JWT), enforces per-user ownership, and computes all derived analytics (dashboards, reports, pulse metrics, heatmap, tips).
- **Persistence tier:** a MySQL 8.4 database server holding all users, categories, transactions, budgets, goals and split bills.

2.1 Request flow

Every protected request carries an `Authorization: Bearer <access token>` header. DRF authenticates the token, the view restricts its queryset to rows owned by the requesting user (ownership mixin), serializers validate and shape the payload, and computed fields are attached on the way out. If an access token has expired, the frontend transparently exchanges the stored refresh token for a new access token and retries the request once, so an active user is never interrupted.

2.2 Repository layout

Path	Responsibility
config/	Django project settings, root URL conf, WSGI/ASGI entry points
accounts/	Custom user model, registration, JWT login/refresh, password reset, profile endpoint
core/	Domain models, serializers, API views, analytics engines, URL routing
frontend/	React SPA: pages, layout, charts, styles, API client, session context
documentation/	This PDF and the narrated walkthrough video

3. Technology Stack

Layer	Technology	Role
Backend framework	Django 6.1 + Django REST Framework	REST API, ORM, migrations, admin
Authentication	SimpleJWT	Access/refresh token issuing and validation
Database	MySQL 8.4 (mysqlclient driver)	Relational persistence on port 3307
Frontend framework	React 18 + React Router	SPA pages and client-side routing
Build tooling	Vite	Dev server, hot reload, bundling
Charts	Hand-rolled SVG components	Bars, donut, duo-bars, heatmap, simulator projection
Styling	Single hand-written stylesheet	Neo-brutalist design system, dark mode, responsive breakpoints
Runtime	Python 3.14, Node.js 24	Server and tooling runtimes

4. Installation and Setup

4.1 Prerequisites

- Python 3.12+ with `venv` available
- Node.js 18+ and npm
- A MySQL 8.x server instance (the project targets a local service on port 3307)

4.2 Backend

- Create and activate a virtual environment, then install dependencies: `pip install -r requirements.txt`
- Create a `.env` file supplying the database connection settings and the Django secret key. Credentials are read from the environment only and are never committed to source control.
- Create the application database and a dedicated database user with rights limited to that schema.
- Apply migrations: `python manage.py migrate` — this builds every table listed in section 5.
- Start the API: `python manage.py runserver` (serves `http://127.0.0.1:8000/api/`).

4.3 Frontend

- From `frontend/`: `npm install` then `npm run dev` (serves `http://127.0.0.1:5173`).
- The API base URL is taken from `VITE_API_URL` and defaults to the local API above.

4.4 First run

Open `http://127.0.0.1:5173`, choose **Create account**, pick your country (this sets your currency symbol site-wide), and sign in. Default categories for income and expense are provisioned automatically for each new user.

Security note. Database credentials and the Django secret key live only in `.env` (and, on this machine, in a local phpMyAdmin config used for manual inspection). Neither file may be published, committed, or shared.

5. Database Schema

All domain tables carry an owning-user foreign key; cascade deletes keep referential integrity when an account is removed.

Table	Key columns	Purpose
<code>user</code> (custom)	username, email, first_name, country, monthly_savings_goal, is_staff	Account, currency choice and savings target
<code>category</code>	user, name, type (income/expense), is_default	User-defined and system-default labels
<code>transaction</code>	user, category, type, amount, description, transaction_date, is_deleted	The ledger; soft-delete supported
<code>budget</code>	user, category, month, limit	Monthly spending guardrail per category
<code>goal</code>	user, name, emoji, target_amount, saved_amount, deadline	Savings jars with pace tracking
<code>splitbill</code>	user, title, amount, bill_date, paid_by	A shared bill header
<code>splitmember</code>	bill, name, is_user, settled, paid	One person on a bill with their recorded payments
<code>insight</code>	user, tip_text, category, pinned, dismissed	Generated smart nudges

announcement	title, body	Site-wide notices shown on the dashboard ticker
--------------	-------------	---

5.1 Derived (non-stored) values

Balances, category totals, budget utilisation, goal progress, heatmap intensities, pulse metrics (average daily burn, days left, broke date) and split shares are computed on read by the API; they are never duplicated in storage, so reports can never drift from the ledger.

6. API Reference

Base URL: `/api`. All endpoints except registration, login, refresh and password reset require a bearer token. List endpoints accept the owner's records only.

Endpoint	Methods	Description
<code>/auth/register/</code>	POST	Create account; provisions default categories
<code>/auth/login/</code>	POST	Exchange credentials for access + refresh tokens
<code>/auth/refresh/</code>	POST	Exchange refresh token for a new access token
<code>/auth/me/</code>	GET	Current profile incl. country and savings goal
<code>/auth/password-reset/</code> , <code>-confirm/</code>	POST	Token-based password recovery
<code>/dashboard/</code>	GET	Month income, expenses, balance, top category, budgets, tips
<code>/pulse/</code>	GET	Health score, average daily burn, active days, days left, broke date
<code>/roommate/</code>	POST	Intent-driven money assistant (see section 8)
<code>/transactions/</code> <code>/<id>/</code>	GET POST / GET PUT DELETE	Ledger entries; soft delete on DELETE
<code>/transactions/import-csv/</code>	POST	Bulk import from CSV with per-row error report
<code>/categories/</code> <code>/<id>/</code>	GET POST / GET PUT DELETE	Personal category labels
<code>/budgets/</code> <code>/<id>/</code>	GET POST / GET PUT DELETE	Monthly limits per category
<code>/goals/</code> <code>/<id>/</code>	GET POST / GET PUT DELETE	Savings jars; deposits via PATCH/PUT of saved_amount
<code>/splits/</code> <code>/<id>/</code>	GET POST / GET DELETE	Shared bills with nested members
<code>/splits/<id>/members/<mid>/settle/</code>	POST	Record a member's payment (or full-share shortcut)
<code>/heatmap/</code>	GET	364 days of daily expense totals for the calendar grid
<code>/reports/monthly/?month=YYYY-MM-DD</code>	GET	Month breakdown: categories, income sources, daily, weekly, six-month swing
<code>/insights/</code> <code>/<id>/</code>	GET / PATCH DELETE	Smart tips; pin or dismiss
<code>/insights/generate/</code>	POST	Regenerate tips from current habits
<code>/announcements/</code>	GET	Dashboard ticker notices
<code>/admin/analytics/</code>	GET	Staff-only campus-wide aggregates

6.1 Error contract

Failures return standard HTTP statuses with a JSON body. The frontend flattens `detail` (string or list) and field-keyed messages into a single human-readable sentence shown inline on the form or as a retry-able error state.

7. Feature Guide — Page by Page

7.1 Landing page (/)

Public marketing front door: hero statement, feature bento grid, three-step path, and a sitemap linking every route in the application. No authentication required.

7.2 Register and sign in (/register , /login , /reset-password)

Registration captures first name, username, email, password and country. The country selection sets the currency symbol used across the entire interface (for example ₦, ¢, ₹, \$, £, €). Sign-in issues a JWT pair; the refresh token is retained so sessions survive access-token expiry. Password reset is token based.

7.3 Dashboard (/dashboard)

- Three headline metrics — money in, money out, still yours — with animated count-up.
- Financial health ring and "broke meter": average burn per day, active logging days, and the projected date the balance hits zero.
- Streak board rewarding consecutive logging days.
- Spending-rhythm bar chart and wallet-split donut for the month, top category, and a budget-vs-actual mini list.
- Announcement ticker and the strongest current smart tip.

7.4 Transactions (/transactions)

The ledger. Add income or expense with category, amount, date, description and optional category hint learning ("campus cafe" suggests Food). Filter by type and month, search text, edit, soft-delete, and bulk-import from CSV with a per-row error report.

7.5 Budgets (/budgets)

Set a monthly limit per expense category. Cards show spent vs limit with a progress track that turns amber at 80% and pulses red when exceeded; the sidebar bell also raises alerts at 80%.

7.6 Goal jars (/goals)

Multiple named savings jars with emoji, target, saved amount and optional deadline. Each jar visualises fill level, percentage, and a daily "drip" figure — how much to set aside per day to land on the deadline. Deposits, edits and deletes are inline.

7.7 Split & settle (/splits)

Shared-bill tracker with partial payments. A bill stores a total and an even per-head share; every member (including "You") records how much they have actually paid. Each row shows share, paid and still-owes; summary tiles aggregate what friends owe you, what you owe, and your net position. A "Full share" button settles a member in one tap. Worked example: a ₦450 rent split three ways is ₦150 per head — if you pay ₦150 you are square, a friend who pays ₦50 still owes ₦100, and a friend who pays nothing owes ₦150.

7.8 Spending heatmap (/heatmap)

A 12-month GitHub-style calendar. Each square is one day, printed with its day-of-month number and shaded by spend intensity (quartiles of the year's busiest day). Hovering reveals the exact date and amount; month and weekday guides frame the grid.

7.9 What-if simulator (/simulator)

Sliders for your top five categories let you cut each by a percentage. The panel contrasts current vs simulated paths in plain language: spend per day, money left in 30 days, and how many days longer your balance lasts, plus a two-line

projection chart and a sentence-form verdict.

7.10 Forecast room (/forecast)

Projects the balance forward at the current average burn, states the run-out date in plain words, and lists the levers (biggest categories) that move the date furthest.

7.11 Reports (/reports)

Monthly rewind with month picker: income/expense/balance metrics, category bars and table, category donut, income sources, daily and weekly breakdowns, and a six-month income-vs-expenses swing chart. A print stylesheet produces a clean "Save as PDF" export with the sidebar removed.

7.12 Smart tips (/tips)

Rule-generated nudges derived from your own trail (overspent categories, savings rate, recurring costs). Tips can be pinned, dismissed, or regenerated.

7.13 Categories (/categories)

Create, rename and delete personal income/expense labels. Defaults are provided at registration and remain available.

7.14 Profile (/profile)

Update first name, country/currency and monthly savings goal (used by the assistant and tips); switch colour mode and text size; sign out.

7.15 Admin studio (/admin , staff only)

Campus-wide aggregates for staff accounts: total users, transaction volume and cross-user summaries. Hidden from non-staff navigation.

8. The Money Roommate Assistant

A chat companion available as a floating button on every page and as a standalone page (/roommate). It is not a canned script: each question is parsed into an intent and answered from live database figures.

Intent (example phrasings)	Answer computed from
balance / "how much do I have"	Live income – expenses for the month
afford / "can I afford X"	Balance vs stated price, with burn context
broke / sapa / "when will I run out"	Average daily burn → days left → calendar date
spending rightly / "am I okay"	Expense-to-income ratio verdict, month-on-month delta, top category share, over-budget names
category spend / "how much on food"	Matched category total for the month
week / last month / where did it go	7-day totals, previous month, top-three categories
owe / split / settle / roommate	Per-member share minus paid across all split bills, split into "they owe you" and "you owe"
goal / jar	Jar count, nearest deadline, progress and daily drip
save / saving	Kept amount vs monthly savings goal with a 20% rule-of-thumb
recurring / subscription	Repeated same-category charges

Unmatched questions fall back to a summary of the month plus example questions, so the assistant never fabricates figures.

9. User Interface, Accessibility and Localisation

- **Design system:** neo-brutalist palette (ink #0e1f18, lime #c8ff16, pink #ff2fb3), heavy borders, offset shadows, monospace figures.
- **Responsive:** breakpoints at 1100 / 820 / 640 / 520 / 460 / 400 px; sidebar collapses to a drawer with scrim; grids switch to single column; charts scale fluidly.
- **Dark mode:** full alternate theme via a root attribute, persisted per device.
- **Text size:** three-step control (A– / A / A+) persisted per device.
- **Motion sensitivity:** count-up and float animations respect `prefers-reduced-motion`.
- **Localisation:** country picker drives currency symbol and formatting everywhere, including charts, chat answers and printed reports.
- **Semantics:** labelled controls, aria-busy skeletons, aria-live friendly error states, keyboard-reachable cards.

10. Security and Authentication

- **JWT sessions:** access token lifetime 24 hours, refresh token 30 days; the frontend refreshes silently and retries once on 401 so active users are not bounced to login.
- **Ownership enforcement:** every list/detail view filters by the requesting user; cross-user access returns 404 rather than leaking existence.
- **Password policy:** minimum length enforced at registration and reset; hashes stored by Django's default hasher.
- **Injection safety:** all persistence goes through the ORM or DRF validators; no raw SQL; CSV import validates type, category and date per row.

- **Secrets hygiene:** database credentials and secret key are environment-sourced; CORS wide-open is flagged in settings as development-only.
- **Staff separation:** campus-wide analytics require `is_staff`.

11. Testing and Verification

- Every feature was exercised end-to-end in a real browser against seeded throwaway accounts, which were deleted afterwards (child rows first, respecting foreign-key protection).
- Assistant answers were verified against hand-computed figures (split obligations, category totals, burn rates, ratios).
- Simulator mathematics verified: a 50% cut on the top category reduced daily burn and extended the run-out horizon consistently across the panel and chart.
- Responsive behaviour checked at each breakpoint by resizing and inspecting computed layout for horizontal overflow.
- Database integrity confirmed after each seeding cycle: only the intended accounts remain.

12. Troubleshooting and Maintenance

Symptom	Cause and remedy
Frontend shows "Cannot reach Campus Coin"	API server not running — start <code>runserver</code> and retry
Blank white page	Compile error in a frontend module — read the Vite overlay/console; typically a missing import
Sent back to login mid-session	Refresh token expired or cleared — sign in again; tokens now last 30 days
Mojibake currency symbols in console/scripts	Windows cp1252 console — run Python with UTF-8 IO encoding
Charts empty on a fresh account	Expected — analytics appear with the first transactions
Schema change needed	Edit models, then <code>makemigrations</code> + <code>migrate</code> ; never edit the DB by hand