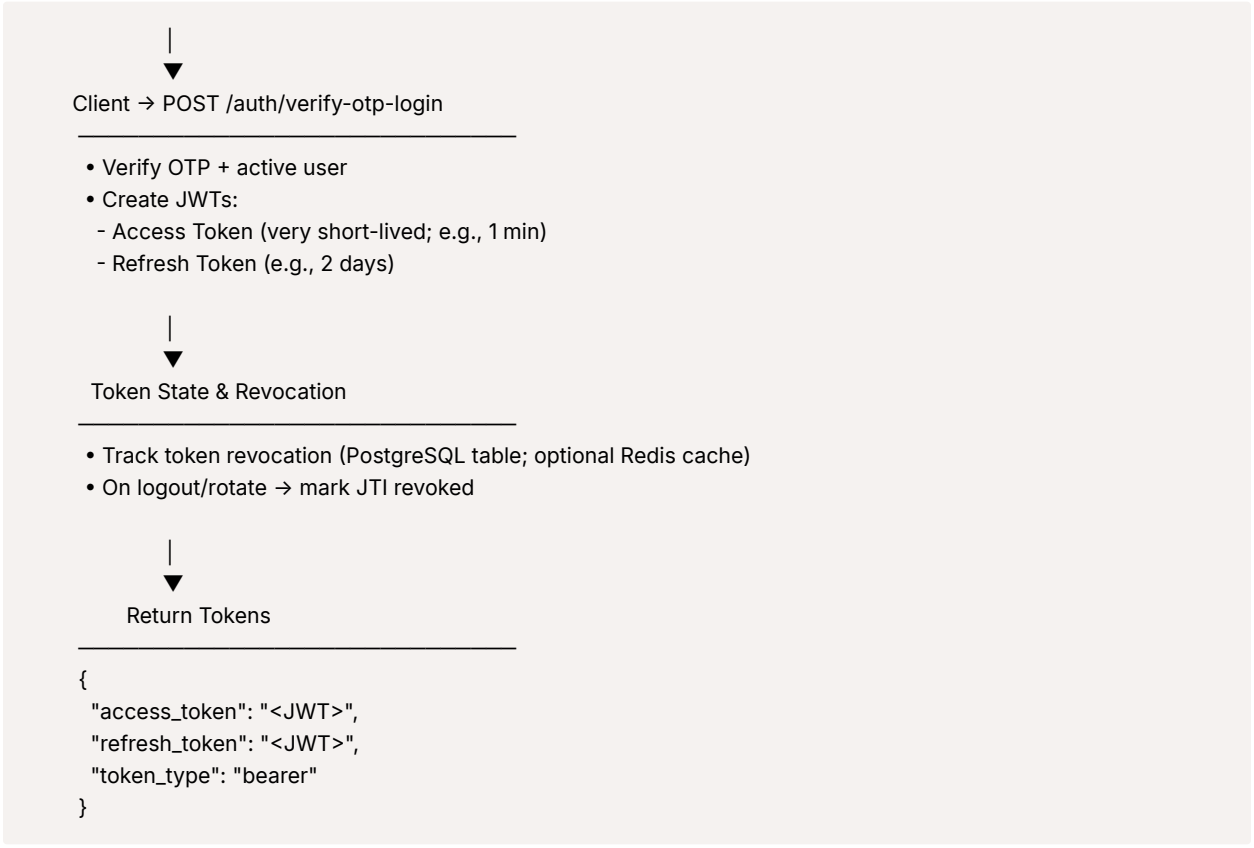
GenchargePhase2-Architecture-Docs



1. Executive Summary

GenCharge is a modern, scalable mobile recharge and digital offers platform built on **FastAPI** with a polyglot backend leveraging **PostgreSQL** (transactional data), **MongoDB** (flexible content & logs), and **Redis** (caching, ephemeral state). It enables seamless OTP-based onboarding, prepaid/postpaid plan browsing, wallet top-ups, subscription management, automated renewals (**Autopay**), offer targeting, referral rewards, and rich transaction history. The system provides robust **role-based access control (RBAC)**, admin consoles for plans, offers, users, content, announcements, backups, and reporting, plus analytics endpoints for operational insights. Real-time-ready architecture, structured audit and backup processes, dynamic notification delivery, and modular domain routers ensure high extensibility and performance for a growing recharge ecosystem.

2. System Overview

2.1 Purpose

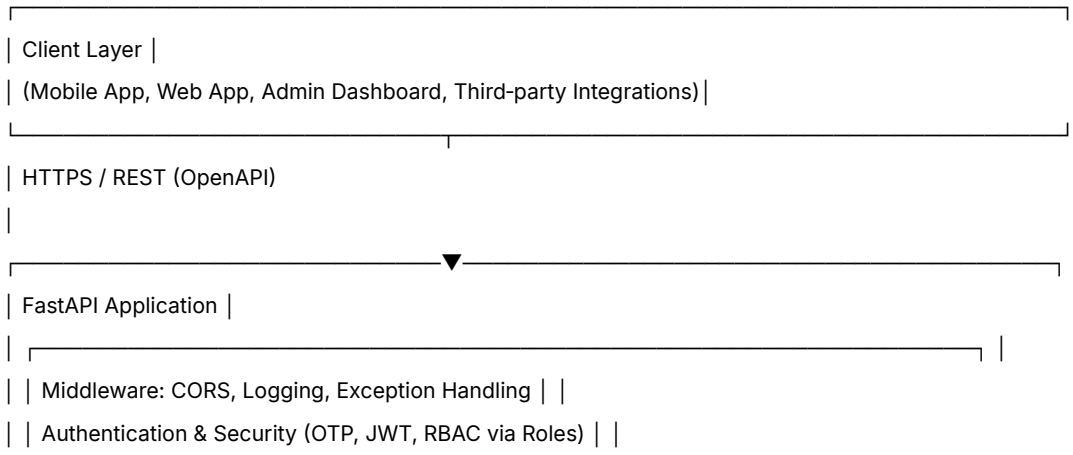
- Secure OTP-based authentication, token refresh, and session lifecycle
- Role-based admin and user management (RBAC: roles, permissions)
- Discovery of mobile recharge plans (grouping, filtering, lifecycle status)
- Offer and promotional management (types, special flags, public vs admin scope)
- Wallet top-up and subscription (recharge flows, active plan tracking)
- Automated recurring payments via Autopay (scheduling, due processing)

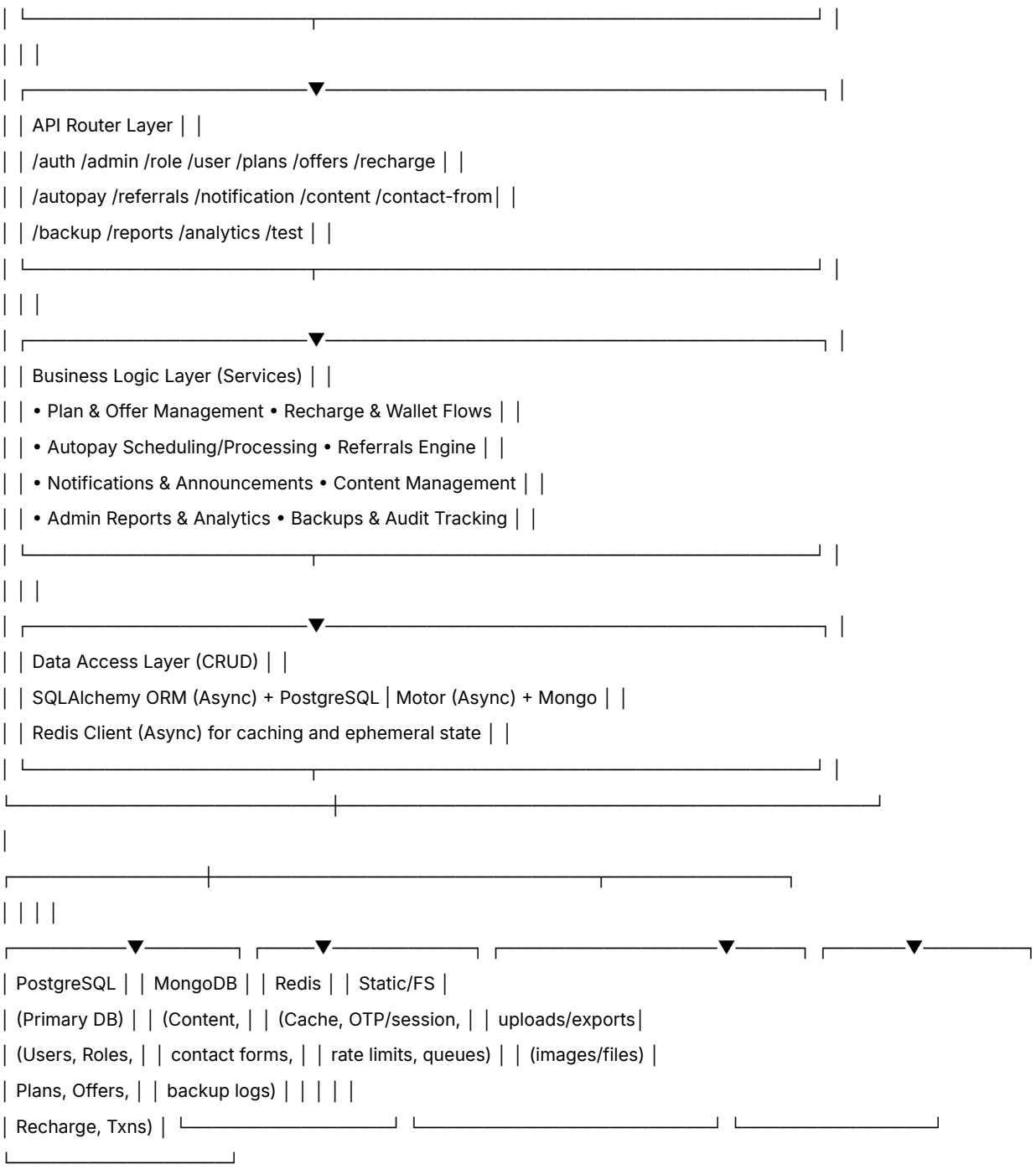
- Referral reward tracking (user/self and admin-wide views)
- Notification delivery and announcement publishing
- User preference management and account lifecycle (activate/deactivate, block/unblock)
- Content management (dynamic informational entries)
- Contact form intake and resolution workflow
- System backups and backup analytics
- Rich transaction and plan analytics plus reporting endpoints
- Centralized audit-style operational visibility (actions via admin endpoints)

2.2 Key Features

Feature	Description
User Authentication	Signup, login, OTP verification, logout, refresh JWT tokens
RBAC & Admin Management	Roles, permissions, admin CRUD, scoped access control
User Lifecycle	Profile info, archive/block/unblock, type switching, deactivate/reactivate
Plan Catalog	Plan groups, plan CRUD, public active plan listing with search & filters
Offers	Offer types, offer CRUD, public offer listing, special & status filtering
Recharge & Wallet	Wallet top-up, subscribe to plans, view active plans with validity filters
Transactions	Detailed user/admin transaction history with extensive query filters
Autopay	Create, edit, list, process due recurring payments for recharge automation
Referrals	Track referral rewards (self and system-wide) with status & pagination
Notifications	User notifications and admin announcements management
Content Management	Admin CRUD for dynamic content entries (stored alongside other data)
Contact Form	Submission intake, listing with filters, mark resolution state
User Preferences	Store and update per-user configurable settings
Reporting	Reports endpoints for operational/financial insights
Analytics	Active plans, transactions, and various user/offer/plan analytic endpoints
Backups	Database backup endpoints and backup analytics tracking
Security & Sessions	Short-lived access tokens, refresh flow, OTP expiry controls
Data Stores	PostgreSQL (core transactional), MongoDB (content/log-style docs), Redis (caching & ephemeral state)

3. High-Level Architecture





4. Component Architecture

4.1 Application Layer Components

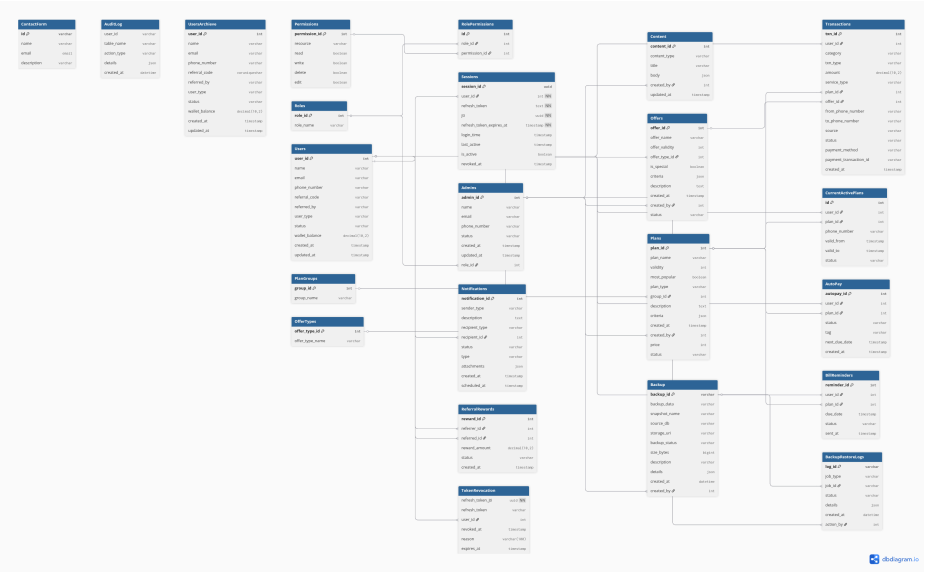
Service	Responsibilities
Auth Service	OTP signup/login, JWT issue/refresh, logout, token revocation controls
Admin Service	Admin self-management, admin CRUD, role assignment
RBAC Service	Roles, permissions, access checks across routers

Service	Responsibilities
User Service	Profile, preferences, lifecycle (deactivate/reactivate), block/unblock, archive views
Plan Service	Plan group CRUD, plan CRUD, public active plan discovery with filters
Offer Service	Offer type CRUD, offer CRUD, public offers listing, special/status flags
Recharge Service	Wallet top-up, plan subscription, active plan history and validity windows
Transaction Service	User/admin transaction queries with rich filters and pagination
Autopay Service	Create/edit/list autopays, process due autopays on schedule
Referral Service	User self referral history, admin-wide referral tracking
Notification Service	Announcements creation, per-user notifications retrieval and deletion
Content Service	Admin content CRUD for dynamic informational entries
Contact Service	Contact form submission, listing with date/email filters, resolve flow
Reports Service	Operational/financial reporting endpoints
Analytics Service	Active plan and transaction analytics, user/offer/plan insights
Backup Service	Backup execution, backup listing and backup analytics visibility
Middleware	CORS, structured logging, unified exception handling, custom docs UI

4.2 Data Layer Components

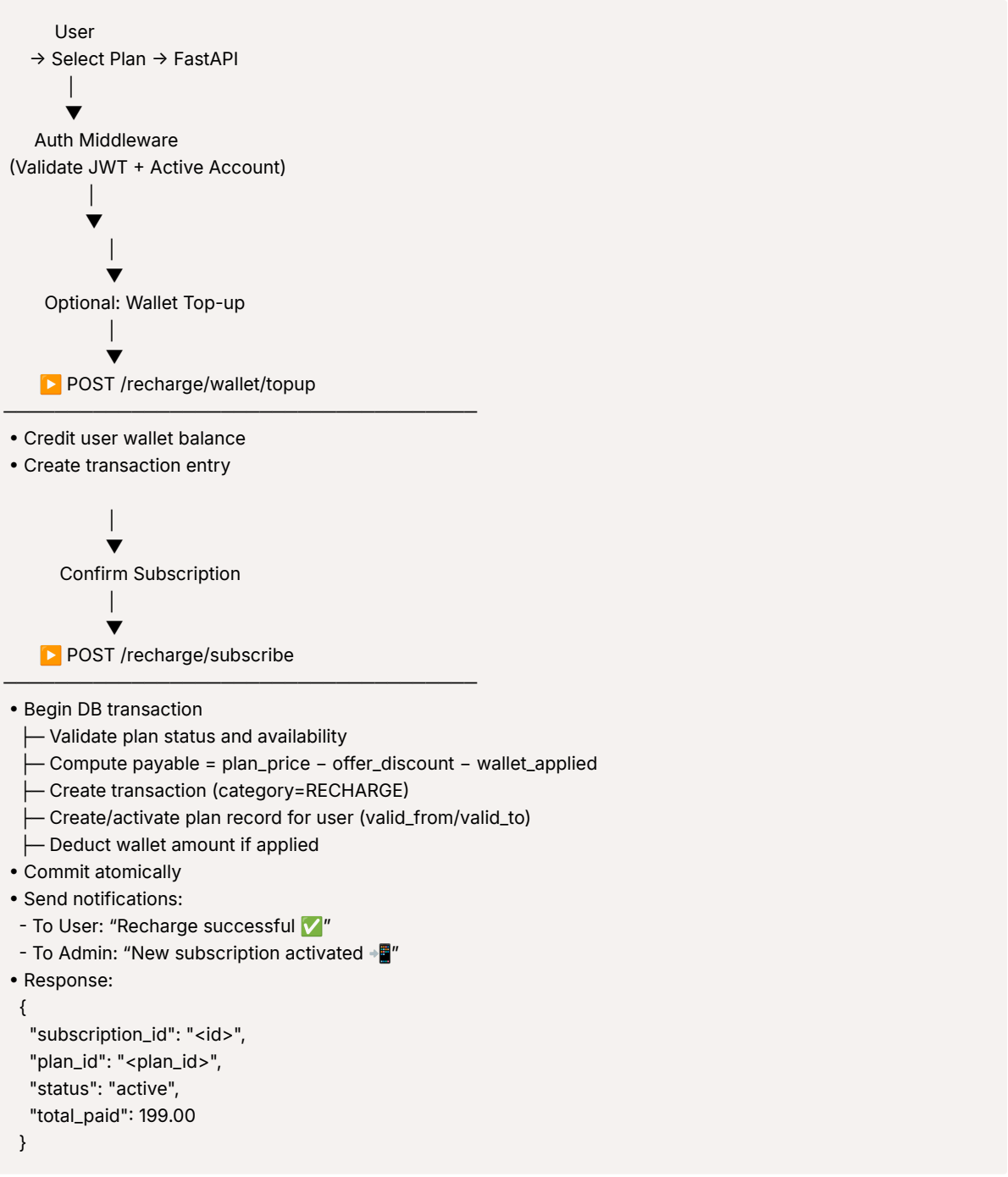
Component	Purpose
PostgreSQL	Primary transactional store: users, roles/permissions, plans, offers, subscriptions, transactions, autopay, reports/analytics aggregates
MongoDB	Flexible document store: content entries, contact-form submissions, backup/audit-style logs
Redis	Ephemeral store: caching, OTP/session data, rate limiting, lightweight queues for due processing
Static/File Storage	Local/static uploads for assets (e.g., images/docs) under uploads

5. Database Schema Overview



6. Data Flow Diagrams

6.1 Recharge Subscription Flow



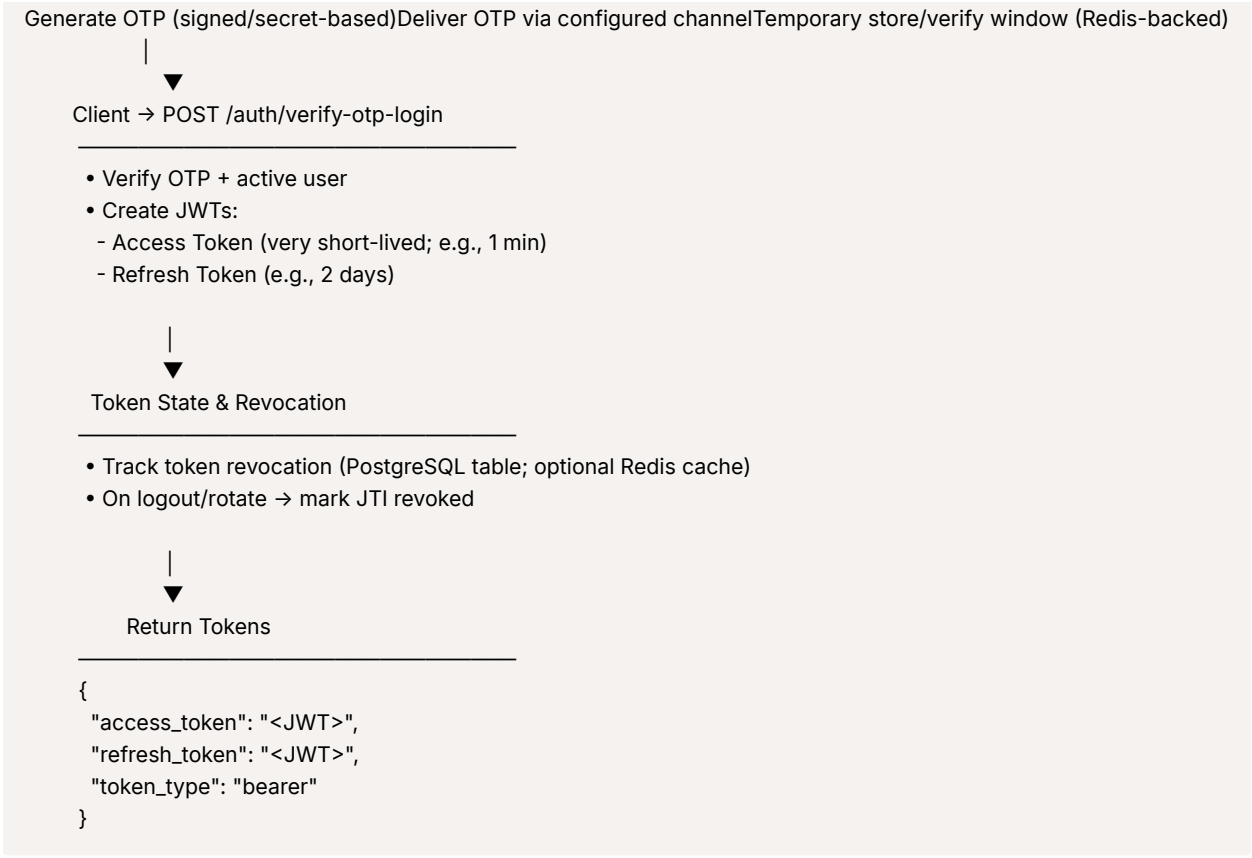
6.2 Admin Plan Update Flow

```
Validate Input (Pydantic)
──────────────────────────────────────────
Validate body against PlanUpdate schemaCheck required fields (name, price, type, status)
                    │
                    ▼
  Update PostgreSQL (SQLAlchemy, async)
──────────────────────────────────────────
  • Begin DB transaction
  • Update  table (fields: price, type, status, metadata)
  • Commit

                    │
                    ▼
     Invalidate Redis Cache
──────────────────────────────────────────
  • Remove keys/patterns:
    - plan:{plan_id}
    - plans:public:* (filters/search variants)

                    │
                    ▼
  Log to MongoDB (Audit/Backup Logs)
──────────────────────────────────────────
  • Insert entry:
    {
      "action": "UPDATE_PLAN",
      "user_role": "admin",
      "plan_id": "<id>",
      "changes": {"price": 199, "status": "ACTIVE"},
      "timestamp": "2025-11-11T20:15:32Z"
    }

                    │
                    ▼
        Return Response
──────────────────────────────────────────
  ✅ HTTP 200 OK
  {
    "message": "Plan updated successfully",
    "plan_id": "<id>",
    "updated_fields": { ... },
    "cache_status": "Invalidated"
  }
```

7. Security Architecture

7.1 Authentication Flow (OTP + JWT)

```
   Client → POST /auth/login (phone_number) → FastAPI Auth
  │
  ▼
  Issue/Verify OTP
  ──────────────────────────────────────
```



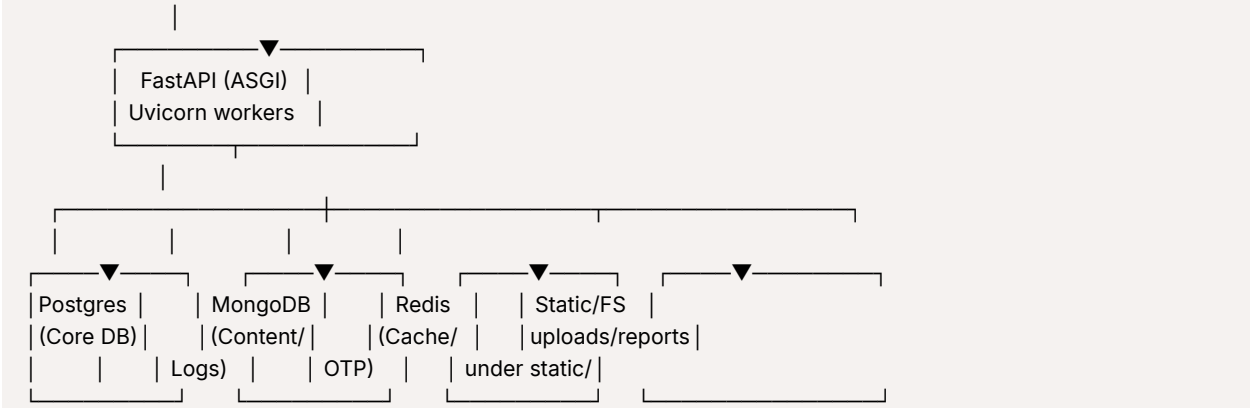
7.2 Authorization Matrix (RBAC)

Role	Users	Plans	Offers	Recharge/Wallet	Autopay	Referrals	Notifications
Admin	Full Access	Full Access	Full Access	View All, Manage	View All, Manage	View All	Announce + Manage
User	Own Profile only	Read public active plans	Read public offers	Own actions & history	Own autopays	Own history	Read own

7.3 Security Measures

- OTP-first authentication with expiry windows and secret-based verification
- JWT access/refresh tokens; revocation tracking by JTI (DB, optionally cached)
- Input validation via Pydantic; strict typing and constraints
- Parameterized queries via SQLAlchemy ORM (SQL injection protection)
- CORS, structured logging, and unified exception middleware
- Principle of least privilege enforced via roles/permissions
- Backup endpoints and logs for recovery/auditing (MongoDB/Files)

8. Deployment Architecture



9. API Architecture Patterns

9.1 RESTful Endpoints (selected)

Method	Endpoint	Purpose
POST	/auth/login	Request OTP for login
POST	/auth/verify-otp-login	Verify OTP and issue JWTs
GET	/plans/public/all	Browse active plans with filters/pagination
POST	/recharge/wallet/topup	Add funds to wallet
POST	/recharge/subscribe	Subscribe to a plan (creates transaction + active plan)
GET	/recharge/my/transactions	View own transactions with rich filters
POST	/autopay/create_autopay	Create recurring recharge rule
POST	/notification/announcement	Admin announcement publish

9.2 Response Standard (typical)

{
"status": "success",
"data": { },
"message": "Operation successful",
"timestamp": "2025-11-11T10:30:00Z",
"request_id": "c0f2d6a1-..."
}

10. Performance & Scalability

Strategy	Implementation Details
Indexing	Add indexes on frequent filters: status, type, created_at, user_id, plan_id.
Caching	Redis cache for public plans/offers lists and common lookups; targeted invalidation on admin updates.
Pagination	Page/size across listing endpoints; support size=0 for full export where provided.
Async I/O	Async SQLAlchemy/Motor for concurrent DB operations; minimize round-trips.
Query Shaping	Project only needed fields for public listings; use server-side sorting.
Batching	Group writes inside single transactions (subscribe, admin bulk edits).

11. Monitoring & Observability

Tool / Method	Purpose
Structured Logging Middleware	Request/response logging with correlation, error traces.
Exception Middleware	Consistent error envelopes and stack capture.
OpenAPI Docs	Discoverability via docs (custom UI) and `/redoc`.
Admin Analytics/Reports	Operational insights exposed via `/analytics` and `/reports`.

12. Future Enhancements

- Payment gateway integration for seamless online payments
- Scheduled worker for automatic Autopay processing
- Push notifications (mobile/web) and optional WebSockets for realtime updates
- Advanced behavioral analytics and cohort dashboards
- Multi-operator support and tenant-aware RBAC
- Optional CDN/object storage for media and backup exports

---

13. Technology Stack Summary

Layer	Technology
Framework	FastAPI (ASGI)
Language	Python 3.x
ORM/DB Access	SQLAlchemy (async) for PostgreSQL; Motor (async) for MongoDB
Databases	PostgreSQL (core transactional), MongoDB (content/audit-style docs)
Cache	Redis (caching, OTP/ephemeral state)
Auth	OTP + JWT (access/refresh with revocation tracking)
Validation/Config	Pydantic, pydantic-settings
Docs	OpenAPI with custom Swagger UI at docs, Redoc at `/redoc`
Static Files	Served from uploads
Migrations	Schema managed via SQLAlchemy metadata; migrations tooling can be added late

14. Development Guidelines

14.1 Code Structure

```
Backend
├── app
│   ├── api
│   │   └── routes
│   │       ├── admin
│   │       │   └── admin_router.py
│   │       ├── analyticas
│   │       │   └── analytics_router.py
│   │       ├── audit_logs
│   │       │   └── audit_logs_router.py
│   │       ├── auth
│   │       │   └── auth_router.py
│   │       ├── autopay
│   │       │   └── autopay_router.py
│   │       ├── backup
```

```
│   │   │   └── backup_router.py
│   │   ├── contact_form
│   │   │   └── contact_form_router.py
│   │   ├── content
│   │   │   └── content_router.py
│   │   ├── init.py
│   │   ├── notification
│   │   │   └── notification_router.py
│   │   ├── offers
│   │   │   └── offers_router.py
│   │   ├── plans
│   │   │   └── plans_router.py
│   │   ├── recharge
│   │   │   └── recharge_router.py
│   │   ├── referrals
│   │   │   └── referrals_router.py
│   │   ├── reports
│   │   │   └── reports_router.py
│   │   ├── roles
│   │   │   └── role_router.py
│   │   ├── testing
│   │   │   └── init.py
│   │   └── users
│   │       └── users_router.py
│   ├── core
│   │   ├── config.py
│   │   ├── database.py
│   │   ├── document_db.py
│   │   ├── init.py
│   │   └── redis_client.py
│   ├── crud
│   │   ├── admin.py
│   │   ├── audit_logs.py
│   │   ├── autopay.py
│   │   ├── backup_analytics.py
│   │   ├── backup.py
│   │   ├── contact_form.py
│   │   ├── content.py
│   │   ├── current_active_plan_analytics.py
│   │   ├── init.py
│   │   ├── notification.py
│   │   ├── offer_analytics.py
│   │   ├── offer.py
│   │   ├── offer_type.py
│   │   ├── permissions.py
│   │   ├── plan_analytics.py
│   │   ├── plans.py
│   │   ├── recharge.py
│   │   ├── referral_analytics.py
│   │   ├── referrals.py
│   │   ├── reports.py
│   │   ├── role_permission.py
│   │   ├── role.py
│   │   ├── sessions.py
```

```
│   │   ├── token_revocation.py
│   │   ├── transaction_analytics.py
│   │   ├── user_insights.py
│   │   ├── users_admin_analytics.py
│   │   ├── users_archieve_analytics.py
│   │   └── users.py
│   ├── dependencies
│   │   ├── auth.py
│   │   ├── init.py
│   │   └── permissions.py
│   ├── init.py
│   ├── main.py
│   ├── middleware
│   │   ├── cors.py
│   │   ├── exception.py
│   │   ├── init.py
│   │   └── logging.py
│   ├── models
│   │   ├── admins.py
│   │   ├── autopay.py
│   │   ├── backup.py
│   │   ├── current_active_plans.py
│   │   ├── init.py
│   │   ├── offers.py
│   │   ├── offer_types.py
│   │   ├── permissions.py
│   │   ├── plan_groups.py
│   │   ├── plans.py
│   │   ├── referral.py
│   │   ├── roles_permissions.py
│   │   ├── roles.py
│   │   ├── sessions.py
│   │   ├── token_revocation.py
│   │   ├── transactions.py
│   │   ├── user_preference.py
│   │   ├── users_archieve.py
│   │   └── users.py
│   ├── schemas
│   │   ├── admin.py
│   │   ├── auth.py
│   │   ├── autopay.py
│   │   ├── backup_analytics.py
│   │   ├── backup.py
│   │   ├── contact_form.py
│   │   ├── content.py
│   │   ├── current_active_plans_analytics.py
│   │   ├── init.py
│   │   ├── notification.py
│   │   ├── offer_analytics.py
│   │   ├── offer_group.py
│   │   ├── offer.py
│   │   ├── permissions.py
│   │   ├── plan_analytics.py
│   │   ├── plan_group.py
```

```
│   │   ├── plans.py
│   │   ├── recharge.py
│   │   ├── referral_analytics.py
│   │   ├── referrals.py
│   │   ├── reports.py
│   │   ├── role_permission.py
│   │   ├── role.py
│   │   ├── transaction_analytics.py
│   │   ├── user_insights.py
│   │   ├── users_admins_analytics.py
│   │   ├── users_archive_analytics.py
│   │   └── users.py
│   ├── services
│   │   ├── auth.py
│   │   ├── autopay.py
│   │   ├── backup_analytics.py
│   │   ├── backup.py
│   │   ├── contact_form.py
│   │   ├── content.py
│   │   ├── current_active_plans_analytics.py
│   │   ├── init.py
│   │   ├── notification.py
│   │   ├── offer_analytics.py
│   │   ├── plan_analytics.py
│   │   ├── recharge.py
│   │   ├── referral_analytics.py
│   │   ├── referral.py
│   │   ├── reports.py
│   │   ├── transactions_analytics.py
│   │   ├── user_insights.py
│   │   ├── user.py
│   │   ├── users_admin_analytics.py
│   │   └── users_archieve_analytics.py
│   └── utils
│       ├── analytics.py
│       ├── content.py
│       ├── init.py
│       ├── messages.py
│       ├── otp.py
│       ├── security.py
│       └── seeding.py
├── backups
│   ├── backup_20251107_160311.sql
│   ├── backup_20251107_162014.sql
│   ├── backup_20251107_163236.sql
│   └── backup_20251108_223219.sql
├── backup_test.sql
├── docs
│   └── API_Documentation.md
├── readme.md
├── requirements.txt
└── static
└── uploads
```

```
├── 206432a4-8baf-4765-a31d-3ae043f589d4.jpg
└── be218d60-6f92-4420-97c9-7e112222e871.jpg
```

14.2 Best Practices

- Dependency injection
- Type hints
- 80%+ test coverage
- API versioning
- Structured logging
- Environment-based config