Indian Mutual Funds Data PWA - Technical Blueprint

Version: 1.0
Date: November 23, 2025
Author: Technical Architecture Team
License: Apache 2.0 (Open Source)

Executive Summary

This blueprint outlines a complete technical architecture for a **public domain, non-proprietary Progressive Web Application (PWA)** focused on Indian mutual fund non-financial transaction and information services. The application targets DIY retail investors seeking transparent access to NAV data, KYC status, portfolio tracking, and fund analytics.

Core Philosophy: Privacy-first, offline-capable, open-source platform leveraging public domain APIs without requiring financial transactions.

1. Use Cases & Feature Set

1.1 Primary Use Cases (MVP)

Non-Financial Information Services

Feature	Description	Data Source	Priority
NAV Tracking	Real-time and historical NAV for all AMFI-registered schemes	AMFI/MFapi.in	P0
Fund Search & Discovery	Search by scheme name, AMC, category, returns	AMFI/MFCentral	P0
Fund Comparison	Side-by-side comparison of 2-5 funds (NAV, returns, ratios)	Computed from AMFI data	P0
Portfolio Tracker	Client-side portfolio with holdings, current value, XIRR	Local storage (IndexedDB fallback)	P0
CAS Upload & Parse	Upload Consolidated Account Statement (PDF/CSV) to auto-populate portfolio	Client-side parsing	P1
KYC Status Check	Check KYC compliance status via PAN (if API available)	MFCentral API (auth required)	P2
SIP Calculator	Calculate SIP returns, maturity amounts, goal planning	Client-side computation	P1

Feature	Description	Data Source	Priority
Fund Analytics	Sharpe ratio, Sortino ratio, rolling returns, risk metrics	Computed from historical NAV	P1

1.2 User Personas

Primary Persona: **DIY Investor Raj**

- Age: 28-45, tech-savvy
- Invests through direct plans via MFCentral, Zerodha, Groww
- Wants: Transparency, offline access, no ads, privacy
- Pain points: Fragmented data across platforms, unreliable apps

2. Data Sources & API Strategy

2.1 Primary Data Sources

Option 1: MFapi.in (Recommended for MVP)

Advantages:

- Free, no authentication required
- Updated 3x daily (2:05 PM, 9:05 PM, 3:09 AM IST)
- JSON responses with complete historical data
- 99.9% uptime with monitoring [1]

Endpoints:

```
// All schemes list
GET https://api.mfapi.in/mf

// Latest NAV for specific scheme
GET https://api.mfapi.in/mf/{scheme_code}

// Historical NAV
GET https://api.mfapi.in/mf/{scheme_code}?start=DD-MM-YYYY&amp;end=DD-MM-YYYY
```

Data Structure:

```
{
  "meta": {
    "fund_house": "HDFC Mutual Fund",
    "scheme_type": "Open Ended Schemes",
    "scheme_category": "Equity Scheme - Large Cap Fund",
    "scheme_code": "119551",
    "scheme_name": "HDFC Index Fund-NIFTY 50 Plan-Direct Plan-Growth"
```

```
  },
  "data": [
    {
      "date": "22-11-2025",
      "nav": "196.234"
    }
  ],
  "status": "SUCCESS"
}
```

Option 2: AMFI Daily NAV File (Backup/Batch Processing)

URL: `https://www.amfiindia.com/spages/NAVAll.txt`

Format: Pipe-delimited text file

```
Scheme Code;ISIN Div Payout/ISIN Growth;ISIN Div Reinvestment;Scheme Name;Net Asset Value
119551;;INF179KA1JK5;HDFC Index Fund-NIFTY 50 Plan-Direct Plan-Growth;196.234;22-Nov-2025
```

Use Case: Nightly batch job to populate local database

Option 3: MFCentral API (Future - Authenticated Services)

Capabilities[2]:

- CAS (Consolidated Account Statement) retrieval
- KYC status validation
- PAN-based portfolio aggregation
- Non-financial transaction requests

Authentication: JWT-based, requires OTP validation

Implementation Priority: Phase 2 (requires user authentication flow)

2.2 Data Storage Strategy

Database Choice: PostgreSQL (Recommended)

Rationale:

- Structured, relational data (schemes, NAV history, portfolio)
- ACID compliance for transaction integrity
- Superior indexing for time-series queries (NAV historical data)[3]
- Better for complex aggregations (XIRR, rolling returns)
- JSON support for flexible scheme metadata

Schema Design:

```
-- Schemes Master Table
CREATE TABLE schemes (
  id SERIAL PRIMARY KEY,
  scheme_code VARCHAR(10) UNIQUE NOT NULL,
  scheme_name VARCHAR(500) NOT NULL,
  amc_name VARCHAR(200),
  scheme_category VARCHAR(200),
  scheme_type VARCHAR(100),
  isin_growth VARCHAR(20),
  isin_dividend VARCHAR(20),
  metadata JSONB, -- Flexible for AMC-specific data
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_scheme_code ON schemes(scheme_code);
CREATE INDEX idx_amc_name ON schemes(amc_name);
CREATE INDEX idx_scheme_category ON schemes(scheme_category);

-- NAV History Table (Time-Series)
CREATE TABLE nav_history (
  id BIGSERIAL PRIMARY KEY,
  scheme_code VARCHAR(10) REFERENCES schemes(scheme_code),
  nav_date DATE NOT NULL,
  nav_value NUMERIC(12, 4) NOT NULL,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  UNIQUE(scheme_code, nav_date)
);

CREATE INDEX idx_nav_scheme_date ON nav_history(scheme_code, nav_date DESC);
CREATE INDEX idx_nav_date ON nav_history(nav_date DESC);

-- User Portfolios (Client-side managed, optional server sync)
CREATE TABLE user_portfolios (
  id SERIAL PRIMARY KEY,
  user_id UUID NOT NULL,
  scheme_code VARCHAR(10) REFERENCES schemes(scheme_code),
  folio_number VARCHAR(50),
  units NUMERIC(15, 4),
  avg_purchase_nav NUMERIC(12, 4),
  purchase_date DATE,
  metadata JSONB,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_user_portfolios ON user_portfolios(user_id);
```

Why NOT MongoDB:

- NAV data is inherently relational (time-series with foreign keys)
- Complex aggregations (rolling returns, Sharpe ratio) easier in SQL
- No need for horizontal sharding at retail investor scale

- ACID guarantees critical for portfolio accuracy [4]

2.3 Caching Strategy (Redis)

Purpose: Reduce database load and API rate limits

Cache Patterns:

```
// Cache Structure
{
  // Latest NAV (TTL: 6 hours)
  "nav:latest:{scheme_code}": { nav: 196.234, date: "22-11-2025" },

  // Historical NAV (TTL: 24 hours, immutable for past dates)
  "nav:history:{scheme_code}:{start_date}:{end_date}": [...],

  // Scheme metadata (TTL: 7 days)
  "scheme:meta:{scheme_code}": {...},

  // All schemes list (TTL: 24 hours)
  "schemes:all": [...]
}
```

Implementation (Express middleware):

```
// middleware/redis-cache.js
import { redisClient } from '../config/redis.js';

export const cacheMiddleware = (keyGenerator, ttl = 3600) =&gt; {
  return async (req, res, next) =&gt; {
    const cacheKey = keyGenerator(req);

    try {
      const cached = await redisClient.get(cacheKey);
      if (cached) {
        return res.json(JSON.parse(cached));
      }
    } catch (error) {
      console.error('Redis read error:', error);
    }

    // Override res.json to cache successful responses
    const originalJson = res.json.bind(res);
    res.json = (data) =&gt; {
      if (res.statusCode === 200) {
        redisClient.setex(cacheKey, ttl, JSON.stringify(data))
          .catch(err =&gt; console.error('Redis write error:', err));
      }
      return originalJson(data);
    };

    next();
```

```
  };
};
```

3. Technology Stack

3.1 Frontend Stack

Component	Technology	Justification
Framework	React 18 + TypeScript	Type safety, hooks, large ecosystem
Build Tool	Vite 5	Fast HMR, native ESM, optimized builds [5]
SSR	Vite SSR Mode	SEO-friendly, faster initial load
State Management	Redux Toolkit	Normalized data, DevTools, predictable updates
UI Components	Tailwind CSS + Shadcn/ui	Utility-first, accessible, customizable
Charts	Recharts	React-native, responsive, declarative
Forms	React Hook Form	Performance, validation, TypeScript support
PWA	Vite Plugin PWA (Workbox)	Service worker, offline caching, install prompt

Project Structure:

```
frontend/
├── src/
│   ├── main.tsx                # Client entry
│   ├── entry-server.tsx        # SSR entry
│   ├── App.tsx
│   ├── components/
│   │   ├── ui/                 # Shadcn components
│   │   ├── FundCard.tsx
│   │   ├── NavChart.tsx
│   │   └── PortfolioTable.tsx
│   ├── features/
│   │   ├── funds/
│   │   │   ├── fundsSlice.ts
│   │   │   ├── FundSearch.tsx
│   │   │   └── FundDetails.tsx
│   │   ├── portfolio/
│   │   │   ├── portfolioSlice.ts
│   │   │   └── Portfolio.tsx
│   │   └── calculators/
│   │       └── SIPCalculator.tsx
│   ├── services/
│   │   └── api.ts              # Axios/Fetch wrapper
│   ├── hooks/
│   │   ├── useOfflineSync.ts
│   │   └── useFundData.ts
│   ├── utils/
│   │   ├── calculations.ts     # XIRR, Sharpe, etc.
│   │   └── formatters.ts
```

```
│   └── sw.ts                    # Service Worker config
├── public/
│   ├── manifest.json
│   └── icons/
├── vite.config.ts
└── package.json
```

3.2 Backend Stack

Component	Technology	Justification
Runtime	Node.js 20 LTS	Active LTS, performance improvements
Framework	Express.js 5	Your expertise, middleware ecosystem
ORM	Prisma 5	Type-safe, migrations, PostgreSQL support
Validation	Zod	Runtime type validation, TypeScript integration
Cache	Redis (ioredis)	High-performance key-value store
Task Queue	BullMQ (Redis-based)	Reliable job processing for background tasks
Cron Jobs	Node-cron	Scheduled NAV updates
API Docs	OpenAPI 3 + Swagger UI	Self-documenting APIs

Project Structure:

```
backend/
├── src/
│   ├── index.ts                 # App entry
│   ├── server.ts                # Express server
│   ├── config/
│   │   ├── database.ts          # Prisma client
│   │   └── redis.ts             # Redis client
│   ├── routes/
│   │   ├── funds.ts
│   │   ├── nav.ts
│   │   └── portfolio.ts
│   ├── controllers/
│   │   ├── FundController.ts
│   │   └── NavController.ts
│   ├── services/
│   │   ├── MFApiService.ts      # MFapi.in integration
│   │   ├── AMFIService.ts       # AMFI file parsing
│   │   └── CalculationService.ts
│   ├── jobs/
│   │   ├── NavSyncJob.ts        # Daily NAV sync
│   │   └── SchemeUpdateJob.ts
│   ├── middleware/
│   │   ├── errorHandler.ts
│   │   ├── rateLimit.ts
│   │   └── cache.ts
│   ├── utils/
│   │   ├── validators.ts
```

```
│   │   └── calculations.ts
│   └── types/
│       └── index.ts
├── prisma/
│   ├── schema.prisma
│   └── migrations/
├── package.json
└── tsconfig.json
```

API Design (RESTful):

```
// GET /api/v1/funds?category=equity&amp;amc=HDFC&amp;limit=20
// Response: Paginated fund list

// GET /api/v1/funds/:schemeCode
// Response: Fund details with latest NAV

// GET /api/v1/funds/:schemeCode/nav?start=2024-01-01&amp;end=2025-11-23
// Response: Historical NAV data

// GET /api/v1/funds/:schemeCode/analytics
// Response: Computed metrics (Sharpe, Sortino, rolling returns)

// POST /api/v1/portfolio/calculate
// Body: { holdings: [...] }
// Response: Current value, XIRR, allocation
```

3.3 DevOps & Deployment (Render.com)

Deployment Architecture

Services on Render:

1. **Web Service (Frontend + Backend)** - Free Tier
   - Type: Node.js Web Service
   - Build: `npm run build` (builds both frontend + backend)
   - Start: `npm run start` (serves SSR React + Express API)
   - Auto-deploy: GitHub `main` branch
   - Instance: Free (spins down after 15 min idle) [6]
2. **PostgreSQL Database** - Free Tier
   - Type: Render Postgres
   - Storage: 1 GB (sufficient for ~5 years NAV data)
   - Auto-backups: Daily
3. **Redis Instance** - Free Tier
   - Type: Render Key-Value

- Memory: 100 MB (adequate for hot cache)

4. **Cron Job (NAV Sync)** - Free Tier
   - Type: Cron Job
   - Schedule: `0 21 * * *` (9 PM IST daily, post-market hours)
   - Command: `npm run sync:nav`
   - Execution: Max 12 hours (sufficient for full AMFI sync) [7]

Render Configuration (`render.yaml`):

```
services:
  # Main Web Service
  - type: web
    name: mf-data-pwa
    env: node
    region: singapore
    plan: free
    buildCommand: npm ci &amp;&amp; npm run build
    startCommand: npm run start
    envVars:
      - key: NODE_ENV
        value: production
      - key: DATABASE_URL
        fromDatabase:
          name: mf-postgres
          property: connectionString
      - key: REDIS_URL
        fromDatabase:
          name: mf-redis
          property: connectionString
      - key: PORT
        value: 10000
    autoDeploy: true
    healthCheckPath: /api/health

  # Daily NAV Sync Cron
  - type: cron
    name: nav-sync-job
    env: node
    schedule: "0 21 * * *"  # 9 PM IST (UTC+5:30 = 15:30 UTC)
    buildCommand: npm ci
    startCommand: npm run sync:nav
    envVars:
      - key: DATABASE_URL
        fromDatabase:
          name: mf-postgres
          property: connectionString
      - key: REDIS_URL
        fromDatabase:
          name: mf-redis
          property: connectionString

databases:
  - name: mf-postgres
```

```
    plan: free
    databaseName: mf_data
    user: mf_user

  - name: mf-redis
    plan: free
```

CI/CD Pipeline (GitHub Actions):

```
# .github/workflows/deploy.yml
name: Deploy to Render

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '20'
      - run: npm ci
      - run: npm run test
      - run: npm run lint

  deploy:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    steps:
      - name: Trigger Render Deployment
        run: |
          curl -X POST ${{ secrets.RENDER_DEPLOY_HOOK }}
```

Handling Free Tier Limitations:

Limitation	Mitigation Strategy
15-min idle spin-down	PWA works offline; first load after wakeup shows cached data
750 free hours/month	= 31.25 days/month; sufficient if service sleeps when unused
Cold start delay	Service worker shows "Loading..." state; pre-cached data renders immediately
No persistent file storage	Use PostgreSQL for all data; no temp file uploads
100 GB bandwidth/month	Aggressive caching (Redis + CDN for static assets)

4. PWA Implementation

4.1 Service Worker Strategy (Workbox)

Caching Strategy:

```
// vite.config.ts
import { VitePWA } from 'vite-plugin-pwa';

export default defineConfig({
  plugins: [
    VitePWA({
      registerType: 'autoUpdate',
      includeAssets: ['favicon.ico', 'robots.txt', 'apple-touch-icon.png'],
      manifest: {
        name: 'MF Data - Indian Mutual Funds Tracker',
        short_name: 'MF Data',
        description: 'Track NAV, analyze funds, manage portfolio - Privacy-first PWA',
        theme_color: '#1e40af',
        background_color: '#ffffff',
        display: 'standalone',
        orientation: 'portrait',
        scope: '/',
        start_url: '/',
        icons: [
          {
            src: 'pwa-192x192.png',
            sizes: '192x192',
            type: 'image/png'
          },
          {
            src: 'pwa-512x512.png',
            sizes: '512x512',
            type: 'image/png',
            purpose: 'any maskable'
          }
        ]
      },
      workbox: {
        runtimeCaching: [
          {
            // API responses
            urlPattern: /^https:\/\/api\.mfdata\.app\/api\/v1\/.*/i,
            handler: 'NetworkFirst',
            options: {
              cacheName: 'api-cache',
              expiration: {
                maxEntries: 100,
                maxAgeSeconds: 60 * 60 * 6 // 6 hours
              },
              cacheableResponse: {
                statuses: [0, 200]
              }
            }
          },
```

```
          {
            // Static assets (fonts, images)
            urlPattern: /\.(?:png|jpg|jpeg|svg|gif|webp|woff2)$/,
            handler: 'CacheFirst',
            options: {
              cacheName: 'static-assets',
              expiration: {
                maxEntries: 50,
                maxAgeSeconds: 60 * 60 * 24 * 30 // 30 days
              }
            }
          }
        ]
      }
    })
  ]
});
```

4.2 Offline Functionality

Offline-Capable Features:

- ✅ Previously viewed fund details (NAV, charts)
- ✅ Portfolio calculations (current value, XIRR)
- ✅ SIP calculator (client-side math)
- ✅ Fund comparison (if funds already cached)
- ❌ Real-time NAV updates (shows last synced timestamp)

Offline Sync Pattern:

```
// hooks/useOfflineSync.ts
import { useEffect, useState } from 'react';

export const useOfflineSync = () =&gt; {
  const [isOnline, setIsOnline] = useState(navigator.onLine);
  const [pendingSync, setPendingSync] = useState&lt;string[]&gt;([]);

  useEffect(() =&gt; {
    const handleOnline = async () =&gt; {
      setIsOnline(true);

      // Sync pending portfolio updates
      if (pendingSync.length &gt; 0) {
        await syncPortfolio(pendingSync);
        setPendingSync([]);
      }

      // Fetch latest NAV for portfolio holdings
      await refreshPortfolioNav();
    };

    const handleOffline = () =&gt; setIsOnline(false);
```

```
    window.addEventListener('online', handleOnline);
    window.addEventListener('offline', handleOffline);

    return () =&gt; {
      window.removeEventListener('online', handleOnline);
      window.removeEventListener('offline', handleOffline);
    };
  }, [pendingSync]);

  return { isOnline, pendingSync, setPendingSync };
};
```

4.3 IndexedDB for Client-Side Storage

Why IndexedDB (not localStorage):

- Larger storage quota (~50 MB minimum, often 100+ MB)
- Asynchronous API (non-blocking)
- Structured data storage (perfect for portfolio holdings)
- Transaction support

Schema (using Dexie.js):

```
// db/index.ts
import Dexie, { Table } from 'dexie';

interface PortfolioHolding {
  id?: number;
  schemeCode: string;
  schemeName: string;
  folioNumber?: string;
  units: number;
  avgPurchaseNav: number;
  purchaseDate: string;
  lastSyncedNav?: number;
  lastSyncedDate?: string;
}

class MFDatabase extends Dexie {
  portfolio!: Table&lt;PortfolioHolding&gt;;
  cachedNavData!: Table&lt;any&gt;;

  constructor() {
    super('MFDataDB');
    this.version(1).stores({
      portfolio: '++id, schemeCode, folioNumber',
      cachedNavData: 'schemeCode, date'
    });
  }
}
```

```
export const db = new MFDatabase();
```

5. Key Algorithms & Calculations

5.1 XIRR Calculation (Portfolio Returns)

XIRR (Extended Internal Rate of Return) - industry standard for mutual fund returns

Implementation (Newton-Raphson method):

```
// utils/calculations.ts
export interface CashFlow {
  date: Date;
  amount: number; // Negative for investments, positive for redemptions
}

export function calculateXIRR(cashflows: CashFlow[]): number {
  const guess = 0.1; // Initial guess: 10% return
  const maxIterations = 100;
  const precision = 0.0000001;

  let rate = guess;

  for (let i = 0; i &lt; maxIterations; i++) {
    const [npv, dnpv] = calculateNPVandDerivative(cashflows, rate);
    const newRate = rate - npv / dnpv;

    if (Math.abs(newRate - rate) &lt; precision) {
      return newRate * 100; // Convert to percentage
    }

    rate = newRate;
  }

  throw new Error('XIRR calculation did not converge');
}

function calculateNPVandDerivative(
  cashflows: CashFlow[],
  rate: number
): [number, number] {
  const firstDate = cashflows[^0].date.getTime();
  let npv = 0;
  let dnpv = 0;

  for (const cf of cashflows) {
    const days = (cf.date.getTime() - firstDate) / (1000 * 60 * 60 * 24);
    const years = days / 365;
    const factor = Math.pow(1 + rate, years);

    npv += cf.amount / factor;
    dnpv -= (years * cf.amount) / (factor * (1 + rate));
```

```
  }

  return [npv, dnpv];
}
```

5.2 Sharpe Ratio (Risk-Adjusted Returns)

Formula: `(Return - Risk-Free Rate) / Standard Deviation`

```
export function calculateSharpeRatio(
  returns: number[], // Monthly/annual returns as decimals
  riskFreeRate: number = 0.065 // 6.5% (typical Indian govt bond rate)
): number {
  const avgReturn = returns.reduce((a, b) =&gt; a + b, 0) / returns.length;
  const variance = returns.reduce((sum, r) =&gt; sum + Math.pow(r - avgReturn, 2), 0) / r
  const stdDev = Math.sqrt(variance);

  return (avgReturn - riskFreeRate) / stdDev;
}
```

5.3 Rolling Returns

Use Case: Show fund consistency (e.g., 3-year rolling returns over 10 years)

```
export function calculateRollingReturns(
  navData: { date: Date; nav: number }[],
  periodYears: number = 3
): { date: Date; return: number }[] {
  const sortedData = navData.sort((a, b) =&gt; a.date.getTime() - b.date.getTime());
  const rollingReturns: { date: Date; return: number }[] = [];

  for (let i = 0; i &lt; sortedData.length; i++) {
    const startDate = sortedData[i].date;
    const endDate = new Date(startDate);
    endDate.setFullYear(endDate.getFullYear() + periodYears);

    const endNav = sortedData.find(d =&gt; d.date &gt;= endDate);
    if (!endNav) break;

    const cagr = Math.pow(endNav.nav / sortedData[i].nav, 1 / periodYears) - 1;
    rollingReturns.push({ date: endDate, return: cagr * 100 });
  }

  return rollingReturns;
}
```

6. Compliance & Legal

6.1 SEBI Compliance Requirements

Mutual Fund App Regulations (Non-Transaction Platforms):

1. **Disclaimer Requirement**:
   - ❗ Must display: *"Mutual fund investments are subject to market risks. Read all scheme-related documents carefully before investing."*
   - Placement: Footer of every page + before any return projections
2. **No Investment Advice**:
   - Cannot provide specific buy/sell recommendations
   - Can show historical data, analytics, comparisons (factual information only)
   - Must not predict future returns
3. **Data Accuracy**:
   - NAV data must be sourced from AMFI-approved sources
   - Display data source and last updated timestamp
   - Clearly mark computed metrics (Sharpe, XIRR) as "calculated based on historical data"
4. **KYC Compliance** (if implementing MFCentral integration)[8]:
   - User consent required before accessing PAN/portfolio data
   - OTP-based verification mandatory
   - Cannot store KYC documents client-side

Implementation:

```
// components/DisclaimerBanner.tsx
export const DisclaimerBanner = () =&gt; (
  <div>
    <div>
      <div>
        &lt;svg className="h-5 w-5 text-yellow-400" /* ... */&gt;&lt;/svg&gt;
      </div>
      <div>
        <p>
          <strong>SEBI Disclaimer:</strong> Mutual fund investments are subject to market
          Read all scheme-related documents carefully before investing.
          Past performance is not indicative of future returns.
        </p>
      </div>
    </div>
  </div>
);
```

6.2 Privacy-First Design

Principles (aligned with TrustVault approach):

1. **Client-Side Portfolio Storage**:
   - Portfolio data stored in IndexedDB (local browser storage)
   - Optional server sync (encrypted, user-controlled)
   - No analytics/tracking scripts (Google Analytics, etc.)
2. **No User Tracking**:
   - No cookies for behavioral tracking
   - No IP logging (except for rate limiting)
   - No third-party ad networks
3. **Open Data Access**:
   - All API endpoints publicly accessible (no auth for public data)
   - Rate limiting to prevent abuse (100 requests/15 min per IP)

Privacy Policy (Required for PWA install):

```
## Data We Collect
- **Portfolio Data**: Stored locally on your device (IndexedDB). We never access this dat
- **API Usage Logs**: Anonymous request counts for rate limiting (no IP storage).

## Data We DON'T Collect
- No personal identifiable information (PII)
- No browsing history or behavioral tracking
- No cookies (except essential session cookies if logged in)

## Third-Party Data
- NAV data sourced from AMFI/MFapi.in (public domain)
- No data sharing with advertisers or data brokers
```

7. Development Phases & Timeline

Phase 1: MVP (4-6 weeks)

Week 1-2: Backend Foundation

- [ ] Setup Express + Prisma + PostgreSQL
- [ ] Implement MFapi.in service layer
- [ ] Database schema + migrations
- [ ] API endpoints (funds list, NAV history, search)
- [ ] Redis caching middleware
- [ ] Deploy to Render (web service + postgres)

Week 3-4: Frontend Core

- [ ] Vite + React + TypeScript setup
- [ ] Redux Toolkit state management
- [ ] Fund search & listing UI
- [ ] Fund details page with NAV chart
- [ ] SIP calculator
- [ ] PWA configuration (service worker, manifest)

Week 5-6: Portfolio & Polish

- [ ] Portfolio tracker (IndexedDB)
- [ ] Portfolio analytics (current value, XIRR)
- [ ] Offline functionality testing
- [ ] SEBI disclaimer integration
- [ ] Responsive design (mobile-first)
- [ ] Performance optimization (Lighthouse >90)

MVP Deliverables:

- ✅ Search & browse all AMFI funds
- ✅ View NAV charts (1 month, 6 months, 1 year, 5 years)
- ✅ Add funds to portfolio (client-side)
- ✅ SIP calculator
- ✅ Offline-capable PWA
- ✅ Deployed to Render free tier

Phase 2: Advanced Features (8-10 weeks)

Weeks 7-10:

- [ ] CAS upload & parsing (PDF/CSV)
- [ ] Fund comparison tool (side-by-side)
- [ ] Advanced analytics (Sharpe, Sortino, rolling returns)
- [ ] Goal-based planning (retirement, education)
- [ ] MFCentral API integration (KYC status, auth flow)
- [ ] Background sync workers (BullMQ)

Weeks 11-14:

- [ ] Historical NAV backfill (5+ years data)
- [ ] Email alerts (NAV milestones, portfolio rebalancing)
- [ ] Export portfolio (PDF/Excel)

- [ ] Dark mode
- [ ] Multi-language (English, Hindi)

Phase 3: Community & Monetization (12+ weeks)

- [ ] Open-source release (Apache 2.0)
- [ ] GitHub Sponsors integration
- [ ] API access for developers (rate-limited free tier)
- [ ] Premium features (TBD - without violating SEBI rules)
- [ ] Mobile apps (React Native/Capacitor)

8. Testing Strategy

8.1 Unit Tests (Jest + Vitest)

Coverage Targets: >80% for critical paths

```
// __tests__/calculations.test.ts
import { calculateXIRR, calculateSharpeRatio } from '../utils/calculations';

describe('XIRR Calculation', () =&gt; {
  it('should calculate correct XIRR for sample portfolio', () =&gt; {
    const cashflows = [
      { date: new Date('2023-01-01'), amount: -10000 },
      { date: new Date('2023-06-01'), amount: -10000 },
      { date: new Date('2024-01-01'), amount: 25000 }
    ];

    const xirr = calculateXIRR(cashflows);
    expect(xirr).toBeCloseTo(22.47, 1); // ~22.5% return
  });
});
```

8.2 Integration Tests (Playwright)

Critical Flows:

- [ ] Search fund → View details → Add to portfolio
- [ ] Offline functionality (network throttling)
- [ ] PWA install prompt

8.3 Performance Budgets

Lighthouse Targets:

- Performance: >90
- Accessibility: >95
- Best Practices: >90
- SEO: >90
- PWA: ✓ (installable)

Load Time Targets:

- First Contentful Paint: <1.5s
- Time to Interactive: <3s (on 3G network)
- Largest Contentful Paint: <2.5s

9. Cost Analysis (Free Tier Viability)

Render Free Tier Limits

Resource	Free Tier Limit	Estimated Usage	Sufficient?
Web Service Hours	750 hours/month	~500 hours (spins down idle)	✓ Yes
PostgreSQL Storage	1 GB	~300 MB (5 years NAV for 5000 funds)	✓ Yes
Redis Memory	100 MB	~50 MB (hot cache)	✓ Yes
Bandwidth	100 GB/month	~20 GB (with caching)	✓ Yes
Cron Jobs	Free (12 hr max)	1 job, ~10 min daily	✓ Yes

Total Cost: **$0/month** (sustainable on free tier for 1-5K users)

Paid Upgrade Triggers:

- >10K active users → Need paid instance ($7/month web service)
- >1 GB DB → Upgrade to paid Postgres ($7/month for 10 GB)
- >100 GB bandwidth → Overage charges ($0.10/GB)

10. GitHub Repository Structure

10.1 Monorepo Strategy

Structure (Turborepo recommended):

```
mf-data-pwa/
├── apps/
│   ├── web/                    # Frontend (Vite + React)
│   └── api/                    # Backend (Express + Prisma)
├── packages/
│   ├── ui/                     # Shared UI components
│   ├── utils/                  # Shared utilities
│   └── types/                  # Shared TypeScript types
├── .github/
│   └── workflows/
│       ├── deploy.yml
│       ├── test.yml
│       └── release.yml
├── docs/
│   ├── API.md
│   ├── CONTRIBUTING.md
│   └── DEPLOYMENT.md
├── LICENSE                     # Apache 2.0
├── README.md
├── package.json
└── turbo.json
```

10.2 README Template

```
# MF Data - Indian Mutual Funds PWA

&gt; Privacy-first, open-source Progressive Web App for tracking Indian mutual fund NAVs,
&gt; portfolio analytics, and goal planning.

## Features
- □ Search &amp;amp; browse 5000+ AMFI-registered funds
- □ Historical NAV charts with technical indicators
- □ Client-side portfolio tracking (privacy-first)
- □ Advanced analytics (XIRR, Sharpe ratio, rolling returns)
- □ Offline-capable PWA (works without internet)
- □ 100% free, no ads, open source

## Tech Stack
- **Frontend**: React 18 + TypeScript + Vite + Redux Toolkit
- **Backend**: Express.js + Prisma + PostgreSQL + Redis
- **Deployment**: Render.com (Free Tier)
- **License**: Apache 2.0

## Getting Started
\`\`\`bash
# Clone the repo
git clone https://github.com/iAn-P1nt0/mf-data-pwa.git

# Install dependencies
npm install
```

```
# Setup environment
cp .env.example .env

# Run database migrations
npm run db:migrate

# Start development servers
npm run dev
\`\`\`

## Contributing
See [CONTRIBUTING.md](./docs/CONTRIBUTING.md)

## License
Apache 2.0 - see [LICENSE](./LICENSE)
```

11. Next Steps & Action Items

Immediate Actions (Next 48 Hours)

1. **Repository Setup**:
   ```
   mkdir mf-data-pwa &amp;amp;&amp;amp; cd mf-data-pwa
   git init
   npm init -y
   npm install -D turbo
   npx create-turbo@latest
   ```
2. **Render Account Configuration**:
   - Connect GitHub account to Render
   - Create PostgreSQL database (free tier)
   - Create Redis instance (free tier)
   - Note down connection strings
3. **API Exploration**:
   ```
   # Test MFapi.in endpoints
   curl https://api.mfapi.in/mf | jq '.[0:5]'
   curl https://api.mfapi.in/mf/119551 | jq '.data[0:5]'
   ```
4. **Database Schema Design**:
   - Create Prisma schema file
   - Run initial migrations locally
   - Seed with sample AMFI data

Week 1 Milestones

- [ ] Express API with health check endpoint deployed to Render
- [ ] PostgreSQL connected with basic schemas
- [ ] MFapi.in service integration tested
- [ ] Redis caching middleware implemented
- [ ] Frontend boilerplate with Vite SSR

Appendix

A. Useful Resources

APIs & Data:

- MFapi.in Docs: https://www.mfapi.in
- AMFI NAV File: https://www.amfiindia.com/spages/NAVAll.txt
- MFCentral API (requires partnership): https://www.mfcentral.com

Libraries:

- Prisma: https://www.prisma.io/docs
- Vite PWA Plugin: https://vite-pwa-org.netlify.app
- Recharts: https://recharts.org
- Dexie.js (IndexedDB): https://dexie.org

Compliance:

- SEBI Mutual Fund Regulations: https://www.sebi.gov.in/legal/regulations/
- AMFI Best Practices: https://www.amfiindia.com

B. Sample Data Structures

Fund List Response:


```
[
  {
    "schemeCode": "119551",
    "schemeName": "HDFC Index Fund-NIFTY 50 Plan-Direct Plan-Growth",
    "schemeCategory": "Equity Scheme - Large Cap Fund",
    "schemeType": "Open Ended Schemes",
    "amcName": "HDFC Mutual Fund",
    "isinGrowth": "INF179KA1JK5",
    "latestNav": {
      "nav": "196.234",
      "date": "22-11-2025"
    }
```

```
    }
  ]
```

Portfolio Analytics Response:

```
{
  "totalInvested": 150000,
  "currentValue": 187450,
  "absoluteReturns": 37450,
  "xirr": 14.23,
  "allocation": [
    { "category": "Large Cap", "percentage": 45.5, "value": 85290 },
    { "category": "Mid Cap", "percentage": 30.2, "value": 56610 },
    { "category": "Debt", "percentage": 24.3, "value": 45550 }
  ],
  "topHoldings": [
    {
      "schemeCode": "119551",
      "schemeName": "HDFC Index Fund-NIFTY 50 Plan",
      "units": 425.5,
      "investedValue": 75000,
      "currentValue": 83509,
      "returns": 11.35
    }
  ]
}
```

References

[1] MFapi.in - Free India Mutual Fund API (https://www.mfapi.in)

[2] MFCentral API Integration Document (https://www.scribd.com/document/749204247)

[3] PostgreSQL vs MongoDB Comparison (https://aws.amazon.com/compare/the-difference-between-mongodb-and-postgresql/)

[4] Postgres vs MongoDB - Complete Comparison (https://www.bytebase.com/blog/postgres-vs-mongodb/)

[5] Vite SSR Documentation (https://vite.dev/guide/ssr)

[6] Render Free Tier Documentation (https://render.com/docs/free)

[7] Render Cron Jobs (https://render.com/docs/cronjobs)

[8] SEBI KYC Standardization (https://www.angelone.in/news/mutual-funds/sebi-proposes-standardised-kyc-process)

Document Version: 1.0
Last Updated: November 23, 2025
Blueprint Status: Ready for Implementation
Estimated MVP Timeline: 4-6 weeks
Total Estimated Effort: 200-250 hours
[9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30]

⁂

1. https://www.amfiindia.com/net-asset-value
2. https://www.amfiindia.com/net-asset-value/nav-history
3. https://www.amfiindia.com
4. https://www.amfiindia.com/research-information
5. https://www.mfapi.in
6. https://www.reddit.com/r/IndiaInvestments/comments/10f3iwy/track_all_mutual_funds_with_only_pan_some_mfs_not/
7. https://twelvedata.com/blog/introducing-mutual-funds-apis
8. https://stackoverflow.com/questions/77493576/extract-daily-mutual-funds-nav-data-from-amfi-website-and-store-it-in-mongodb
9. https://tarrakki.com/product/api-centre/mutualfunds
10. https://www.mongodb.com/resources/compare/mongodb-postgresql
11. https://aws.amazon.com/compare/the-difference-between-mongodb-and-postgresql/
12. https://estuary.dev/blog/postgresql-vs-mongodb/
13. https://www.sevensquaretech.com/mongodb-vs-postgresql/
14. https://www.integrate.io/blog/mongodb-vs-postgresql/
15. https://vite.dev/guide/ssr
16. https://semaphore.io/blog/nodejs-caching-layer-redis
17. https://www.bytebase.com/blog/postgres-vs-mongodb/
18. https://suhaasya.hashnode.dev/building-a-progressive-web-app-with-react-and-vite
19. https://www.linkedin.com/posts/kiran-d-5b0b84257_nodejs-redis-prisma-activity-7385513805277147136-lVuN
20. https://render.com/docs/cronjobs
21. https://render.com/docs/background-workers
22. https://render.com/docs/free
23. https://render.com/pricing
24. https://schedo.dev/render
25. https://www.angelone.in/news/mutual-funds/sebi-proposes-standardised-kyc-process-for-new-mutual-fund-folios
26. https://ijirem.org/DOC/6-Predictive-Analytics-and-Portfolio-Optimization-A-Study-on-Mutual-Fund-Asset-Allocation-and-Risk-Mitigation.pdf
27. https://stackoverflow.com/questions/75340700/prevent-render-server-from-sleeping
28. https://ksandk.com/newsletter/sebi-skin-in-game-requirements-mutual-funds/
29. https://www.lionbloggertech.com/building-a-comprehensive-indian-mutual-fund-analyzer-with-python-and-streamlit/
30. https://www.scribd.com/document/749204247/MFCentral-API-Integration-Document-CAS-v2-2-1-2