Technical Requirements Document

1. System Architecture

The application follows a client-server architecture running locally on the user's machine:

- Frontend (Client): React JS Single Page Application running on Node.js development server (http://localhost:3000)
- Backend (Server): Python FastAPI REST API server (http://localhost:8000)
- Database: SQLite file (`finance.db`) stored locally
- ML Models: Serialized `.pkl` files stored in `/models` directory

Architecture Flow:

```
User Browser → React Frontend (Port 3000) → FastAPI Backend (Port 8000) → SQLite Database
                                                ↓
                                          ML Model Files
```

2. Technology Stack

Component	Technology	Framework/Library	Purpose
Frontend	JavaScript	React JS	User interface components
Frontend Routing		React Router DOM	Page navigation
Frontend State		React Query (TanStack Query)	API data management
UI Library	CSS	Material-UI (MUI)	Pre-built UI components
API Client	JavaScript	Axios	HTTP requests to backend
Backend	Python	FastAPI	REST API server
Database ORM	Python	SQLAlchemy	Database operations

Database	SQL	SQLite 3	Local data storage
ML Framework	Python	Scikit-learn	Machine learning models
Data Processing	Python	Pandas	Data manipulation
Time Series	Python	Statsmodels	Forecasting models

3. Project Structure

```text
personal-finance-tracker/
├── backend/
│   ├── app/
│   │   ├── models/           # SQLAlchemy database models
│   │   │   ├── account.py
│   │   │   ├── category.py
│   │   │   ├── transaction.py
│   │   │   └── budget.py
│   │   ├── api/              # API route handlers
│   │   │   ├── accounts.py
│   │   │   ├── transactions.py
│   │   │   ├── budgets.py
│   │   │   └── ml_routes.py
│   │   ├── ml/               # Machine learning code
│   │   │   ├── categorization.py
│   │   │   ├── anomaly_detection.py
│   │   │   ├── forecasting.py
│   │   │   └── budget_recommendation.py
│   │   ├── main.py           # FastAPI application
│   │   ├── database.py       # Database connection
│   │   └── config.py         # Configuration settings
│   ├── models/               # Trained ML model files
│   │   ├── category_classifier.pkl
│   │   ├── anomaly_detector.pkl
│   │   └── forecast_model.pkl
│   ├── scripts/
│   │   ├── create_tables.py  # Database setup
│   │   └── seed_data.py      # Default categories
│   ├── requirements.txt      # Python dependencies
│   └── finance.db            # SQLite database (auto-created)
├── frontend/
│   ├── src/
│   │   ├── components/       # Reusable UI components
│   │   │   ├── TransactionForm.js
│   │   │   ├── AccountCard.js
```

```
│   │   ├── BudgetProgress.js
│   │   └── AnomalyAlert.js
│   ├── features/          # Feature-based components
│   │   ├── dashboard/
│   │   ├── transactions/
│   │   ├── accounts/
│   │   ├── budgeting/
│   │   └── reports/
│   ├── services/          # API communication
│   │   └── api.js
│   ├── hooks/             # Custom React hooks
│   │   └── useCategorySuggestions.js
│   ├── App.js
│   └── index.js
└── package.json           # Node.js dependencies

└── README.md              # Setup instructions
```

4. Database Schema

4.1 Complete SQL Table Definitions

```sql
-- Enable foreign key constraints
PRAGMA foreign_keys = ON;

-- Categories table
CREATE TABLE categories (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL UNIQUE,
    type TEXT NOT NULL CHECK (type IN ('INCOME', 'EXPENSE')),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Accounts table
CREATE TABLE accounts (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    type TEXT NOT NULL CHECK (type IN ('CHECKING', 'SAVINGS', 'CREDIT_CARD')),
    initial_balance REAL NOT NULL DEFAULT 0.0,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Transactions table
CREATE TABLE transactions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
    description TEXT,
    amount REAL NOT NULL,
    date TEXT NOT NULL,  -- ISO 8601 format (YYYY-MM-DD)
    transaction_type TEXT NOT NULL CHECK (transaction_type IN ('INCOME', 'EXPENSE',
'TRANSFER')),
    account_id INTEGER NOT NULL,
    category_id INTEGER NOT NULL,
    transfer_id TEXT,    -- UUID to link transfer pairs
    is_anomaly BOOLEAN DEFAULT FALSE,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (account_id) REFERENCES accounts (id) ON DELETE RESTRICT,
    FOREIGN KEY (category_id) REFERENCES categories (id) ON DELETE RESTRICT
);

-- Budgets table
CREATE TABLE budgets (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    category_id INTEGER NOT NULL,
    month INTEGER NOT NULL CHECK (month >= 1 AND month <= 12),
    year INTEGER NOT NULL,
    amount REAL NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (category_id) REFERENCES categories (id) ON DELETE CASCADE,
    UNIQUE(category_id, month, year)

);
```

4.2 Database Indexes for Performance

```
sql
```

```
CREATE INDEX idx_transactions_date ON transactions(date);
CREATE INDEX idx_transactions_account_id ON transactions(account_id);
CREATE INDEX idx_transactions_category_id ON transactions(category_id);

CREATE INDEX idx_budgets_month_year ON budgets(month, year);
```

4.3 Default Data

```

-- Insert default categories on first run
INSERT INTO categories (name, type) VALUES
('Salary', 'INCOME'),
('Investment', 'INCOME'),
('Food & Dining', 'EXPENSE'),
```

```
('Groceries', 'EXPENSE'),
('Rent', 'EXPENSE'),
('Transportation', 'EXPENSE'),
('Entertainment', 'EXPENSE'),
('Healthcare', 'EXPENSE'),

('Shopping', 'EXPENSE');
```

5. API Endpoints Specification

5.1 Core Endpoints

Method	Endpoint	Description	Request Body	Success Response
GET	`/api/accounts`	Get all accounts	-	`{data: [Account]}`
POST	`/api/accounts`	Create new account	`{name, type, initial_balance}`	`{data: Account}`
GET	`/api/transactions`	Get transactions	`?account_id=1&start_date=2024-10-01`	`{data: [Transaction]}`
POST	`/api/transactions`	Add transaction	`{description, amount, date, account_id, category_id, transaction_type}`	`{data: Transaction}`
PUT	`/api/transactions/{id}`	Update transaction	`{description, amount, category_id}`	`{data: Transaction}`
DELETE	`/api/transactions`	Delete transaction	-	`{message: "Deleted"}`

	`/{id}`			
POST	`/api/transfers`	Transfer between accounts	`{from_account_id, to_account_id, amount, date, description}`	`{data: {from_transaction, to_transaction}}`
GET	`/api/categories`	Get all categories	-	`{data: [Category]}`
GET	`/api/budgets`	Get budgets	`?month=10&year=2024`	`{data: [Budget]}`
POST	`/api/budgets`	Set/update budget	`{category_id, month, year, amount}`	`{data: Budget}`
GET	`/api/reports/monthly_spending`	Spending report	`?month=10&year=2024`	`{data: {category: amount}}`

5.2 ML Endpoints

Method	Endpoint	Description	Request Body	Response
POST	`/api/ml/suggest-category`	Suggest category	`{description: "Swiggy Order"}`	`{data: {category_id: 3, confidence: 0.92}}`
GET	`/api/ml/anomalies`	Get anomaly transactions	-	`{data: [Transaction]}`

GET	`/api/ml/forecast`	Get cash flow forecast	-	`{data: {dates: [], balances: []}}`
POST	`/api/ml/retrain`	Retrain ML models	-	`{message: "Retraining started"}`

5.3 Standard Response Format

Success Response:

```
json
```

```
{
  "data": {...},
  "message": "Operation completed successfully",
  "timestamp": "2024-10-16T10:00:00Z"

}
```

Error Response:

```
json
```

```
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Amount must be positive",
    "details": {"amount": ["Must be greater than 0"]},
    "timestamp": "2024-10-16T10:00:00Z"
  }

}
```

6. Frontend Implementation

6.1 Key Pages and Components

Dashboard Page (`/`)

- Display total balance across all accounts
- Show recent transactions (last 10)
- Budget progress bars for each category
- Anomaly alerts for suspicious transactions

- Cash flow forecast chart

Transactions Page (`/transactions`)

- Filterable table of all transactions
- Search by description, filter by date range, account, category
- "Add Transaction" button with form
- Highlight anomalies with warning icons

Add Transaction Form

- Real-time category suggestion as user types description
- 300ms debounce on description input
- Display suggested category as MUI Chip component
- Form validation for amount, date, required fields

Budget Page (`/budget`)

- Monthly budget setup for each category
- Show recommended amounts based on past spending
- Progress visualization for current month
- Previous month comparison

6.2 Frontend Code Examples

API Service Layer (`services/api.js`)

```
import axios from 'axios';

const API_BASE = 'http://localhost:8000/api';

// Configure axios instance
const apiClient = axios.create({
  baseURL: API_BASE,
  timeout: 10000,
});

// API methods
export const transactionAPI = {
  getAll: (filters = {}) => apiClient.get('/transactions', { params: filters }),
  create: (transactionData) => apiClient.post('/transactions', transactionData),
  update: (id, updates) => apiClient.put(`/transactions/${id}`, updates),
  delete: (id) => apiClient.delete(`/transactions/${id}`),
  suggestCategory: (description) => apiClient.post('/ml/suggest-category',
{ description }),
};
```

```
export const accountAPI = {
  getAll: () => apiClient.get('/accounts'),
  create: (accountData) => apiClient.post('/accounts', accountData),
};

export const mlAPI = {
  getAnomalies: () => apiClient.get('/ml/anomalies'),
  getForecast: () => apiClient.get('/ml/forecast'),
  retrainModels: () => apiClient.post('/ml/retrain'),

};
```

Transaction Form with Category Suggestion

```
import React, { useState, useEffect } from 'react';
import { transactionAPI } from '../services/api';

function TransactionForm({ onSubmit, onCancel }) {
  const [formData, setFormData] = useState({
    description: '',
    amount: '',
    date: new Date().toISOString().split('T')[0],
    account_id: '',
    category_id: '',
    transaction_type: 'EXPENSE'
  });
  const [suggestedCategory, setSuggestedCategory] = useState(null);
  const [isLoadingSuggestion, setIsLoadingSuggestion] = useState(false);

  // Debounced category suggestion
  useEffect(() => {
    if (formData.description.length < 3) {
      setSuggestedCategory(null);
      return;
    }

    const timeoutId = setTimeout(async () => {
      setIsLoadingSuggestion(true);
      try {
        const response = await transactionAPI.suggestCategory(formData.description);
        setSuggestedCategory(response.data.data);
      } catch (error) {
        console.error('Failed to get category suggestion:', error);
      } finally {
        setIsLoadingSuggestion(false);
```

```
      }
    }, 300);

    return () => clearTimeout(timeoutId);
  }, [formData.description]);

  const handleSubmit = (e) => {
    e.preventDefault();
    onSubmit(formData);
  };

  return (
    <form onSubmit={handleSubmit}>
      {/* Form fields implementation */}
      {suggestedCategory && (
        <div className="suggestion-banner">
          Suggested category: {suggestedCategory.name}
          <button
            type="button"
            onClick={() => setFormData({...formData, category_id:
suggestedCategory.id})}
          >
            Use This
          </button>
        </div>
      )}
    </form>
  );
}
```

7. AI/ML Subsystem

7.1 Smart Categorization

Model: TF-IDF Vectorizer + Logistic Regression (Scikit-learn)

Training Data:

- Features: Transaction descriptions
- Labels: Category IDs
- Source: User's transaction history + initial synthetic data

Implementation:

```python
# Backend ML code structure
class CategoryClassifier:
    def train(self, transactions_df):
        # Preprocess descriptions
        # Train TF-IDF + Logistic Regression
        # Save model to .pkl file

    def predict(self, description):
        # Load trained model
        # Transform description using TF-IDF

        # Return top category with confidence score
```

7.2 Anomaly Detection

Model: Isolation Forest (Scikit-learn) per category

Features:

- Transaction amount
- Day of week
- Day of month
- Historical spending patterns for the category

Threshold: Flag transactions >3 standard deviations from category mean

7.3 Cash Flow Forecasting

Model: ARIMA (Statsmodels) on daily balance history

Requirements:

- Minimum 45 days of historical data
- Trained on rolling 90-day window

Output: 30-day balance projections with confidence intervals

7.4 Budget Recommendations

Method: Statistical aggregation using Pandas

Logic:

- Calculate average spending for each category over last 3 months
- Exclude anomalous transactions from calculation
- Provide mean and median as reference

8. Setup and Development Guide

8.1 Backend Setup

1. Navigate to backend directory:
2. `cd backend`
3. Install Python dependencies:
4. `pip install -r requirements.txt`
5. Initialize database:
6. `python scripts/create_tables.py`
7. `python scripts/seed_data.py`
8. Start the backend server:
9. `python -m uvicorn app.main:app --reload --port 8000`

8.2 Frontend Setup

1. Navigate to frontend directory:
2. `cd frontend`
3. Install Node.js dependencies:
4. `npm install`
5. Start the development server:
6. `npm start`

8.3 Access Points

- Frontend Application: http://localhost:3000
- Backend API Documentation: http://localhost:8000/docs
- Backend API Server: http://localhost:8000

9. Non-Functional Requirements

9.1 Performance

- Dashboard load time: < 3 seconds with 10,000 transactions
- API response time: < 500ms for 95% of requests
- Category suggestion response: < 300ms

9.2 Usability

- Key actions achievable in 3 clicks or less from dashboard

- Intuitive navigation with clear visual hierarchy
- Responsive design for desktop and tablet

9.3 Security

- Backend API only accessible from localhost
- Input validation on all API endpoints
- SQL injection prevention via SQLAlchemy ORM

9.4 Error Handling

- User-friendly error messages for invalid inputs
- Graceful degradation when ML features unavailable
- Comprehensive logging for debugging

10. Testing Strategy

10.1 Backend Testing

- Unit tests for business logic and ML models
- API integration tests using FastAPI TestClient
- Database transaction rollback in tests

10.2 Frontend Testing

- Component unit tests with React Testing Library
- User flow integration tests
- API mocking for isolated component testing

10.3 ML Model Testing

- Cross-validation during model training
- Prediction accuracy monitoring
- Performance benchmarking

11. Deployment and Operations

11.1 Running in Production

```
# Terminal 1 - Backend
```

```
cd backend
python -m uvicorn app.main:app --port 8000

# Terminal 2 - Frontend
cd frontend
npm run build

npx serve -s build -p 3000
```

11.2 Data Backup

- Regular backup of `finance.db` SQLite file
- Backup of `models/` directory containing trained ML models
- Document backup location and procedure for users

12. Troubleshooting Guide

Common Issues and Solutions

Problem: Frontend cannot connect to backend
Solution:

- Ensure backend is running on port 8000
- Check CORS configuration in FastAPI app
- Verify no firewall blocking localhost connections

Problem: ML features not working
Solution:

- Check model files exist in `backend/models/`
- Verify sufficient training data exists
- Check backend logs for ML-related errors

Problem: Database errors
Solution:

- Delete corrupted `finance.db` file
- Re-run database initialization scripts
- Check database file permissions

Problem: Category suggestions inaccurate
Solution:

- Retrain models with `POST /api/ml/retrain`
- Ensure adequate transaction history exists
- Manually correct mis-categorized transactions