REST API Documentation

Base URL

```
http://localhost:8000
```

Authentication

All endpoints require Basic Authentication.

Credentials

- Username: admin
- Password: password123

Header Format

```
Authorization: Basic YWRtaW46cGFzc3dvcmQxMjM=
```

The value after "Basic" is base64 encoding of username:password.

Endpoints

1. List All Transactions

Retrieve a list of all transactions, with optional filtering.

Endpoint: `GET /transactions`

Query Parameters (Optional):

- `type` - Filter by transaction type (payment, transfer, received, deposit, airtime)
- `amount_min` - Minimum transaction amount
- `amount_max` - Maximum transaction amount
- `sender` - Filter by sender name (partial match)
- `recipient` - Filter by recipient name (partial match)

Request Example:

```
curl -u admin:password123 http://localhost:8000/transactions
```

Request with Filters:

```
curl -u admin:password123
"http://localhost:8000/transactions?type=payment&amount_min=1000"
```

Response Example (200 OK):

```
{   "success": true,   "count": 50,   "total": 1693,   "filters": {
"type": ["payment"],     "amount_min": ["1000"]   },   "transactions": [...]
}
```

2. Get Specific Transaction

Retrieve details of a single transaction by ID.

Endpoint: `GET /transactions/{id}`

URL Parameters:

- `id` (required) - Transaction ID (integer)

Request Example:

```
curl -u admin:password123 http://localhost:8000/transactions/1
```

Response Example (200 OK):


```
{   "success": true,   "transaction": {     "id": 1,     "transaction_id":
"76662021700",     "type": "received",     "amount": 2000,     "sender":
"Jane Smith",     "recipient": null,     "phone_number": "*********013",
"fee": null,     "new_balance": 2000   } }
```

3. Create New Transaction

Add a new transaction to the database.

Endpoint: `POST /transactions`

Request Headers:

```
Content-Type: application/json Authorization: Basic YWRtaW46cGFzc3dvcmQxMjM=
```

Request Body:

```
{   "type": "payment",   "amount": 5000,   "recipient": "John Doe",
"sender": null,   "phone_number": "250788123456",   "fee": 100,
"new_balance": 45000,   "readable_date": "02 Feb 2026 10:30:00 AM" }
```

Required Fields:

- `type` - Transaction type (string)

Optional Fields:

- `amount` - Amount in RWF (integer)
- `recipient` - Recipient name (string)
- `sender` - Sender name (string)
- `phone_number` - Phone number (string)
- `fee` - Transaction fee (integer)
- `new_balance` - Balance after transaction (integer)

Response Example (201 Created):

```
{   "success": true,   "message": "Transaction created successfully",
"transaction": {     "id": 1694,     "type": "payment",     "amount": 5000,
"recipient": "John Doe",     "fee": 100,     "new_balance": 45000   } }
```

4. Update Transaction

Modify an existing transaction.

Endpoint: `PUT /transactions/{id}`

URL Parameters:

- `id` (required) - Transaction ID (integer)

Include only the fields you want to update in the request body.

Request Example:

```
curl -u admin:password123 -X PUT http://localhost:8000/transactions/1 \   -H "Content-Type: application/json" \   -d '{"amount": 7500, "fee": 150}'
```

Response Example (200 OK):

```
{   "success": true,   "message": "Transaction updated successfully", "transaction": {     "id": 1,     "amount": 7500,     "fee": 150   } }
```

5. Delete Transaction

Remove a transaction from the database.

Endpoint: `DELETE /transactions/{id}`

URL Parameters:

- `id` (required) - Transaction ID (integer)

Request Example:

```
curl -u admin:password123 -X DELETE http://localhost:8000/transactions/1
```

Response Example (200 OK):

```
{   "success": true,   "message": "Transaction 1 deleted successfully" }
```

Error Codes

Status Code	Description
200	OK - Request successful
201	Created - Resource created successfully
400	Bad Request - Invalid request format or missing required fields
401	Unauthorized - Authentication required or credentials invalid
404	Not Found - Resource does not exist
500	Internal Server Error - Server error occurred

Security Considerations

Basic Authentication Limitations

Understanding the weaknesses of Basic Authentication:

1. Not Encrypted
Credentials are only base64-encoded, not encrypted. Anyone intercepting the request can easily decode and steal credentials. For example, YWRtaW46cGFzc3dvcmQxMjM= decodes to admin:password123.

2. Credentials Sent with Every Request
Username and password are transmitted repeatedly, creating more opportunities for interception.

3. No Expiration
Credentials remain valid indefinitely unless manually changed. Stolen credentials can be used forever.

4. No Rate Limiting
Without rate limiting, attackers can easily brute force credentials by trying unlimited password combinations.

5. Hardcoded Credentials
In this implementation, credentials are in source code, which is a major security vulnerability in production environments.

Recommended Alternatives

1. JWT (JSON Web Tokens)
Token-based authentication that can expire after a set time, contains encoded user information, and can be revoked when needed.

```
POST /auth/login → Returns JWT token GET /transactions (with JWT in header) → Access granted
```

2. OAuth 2.0
Industry standard authentication that eliminates password sharing, provides scope-based permissions, and supports refresh tokens. Users can authenticate through trusted third-party providers like Google or GitHub.

3. API Keys
Easy to implement and can be rotated as needed. Different keys can be generated for different clients or applications.

4. HTTPS with Basic Auth
If you must use Basic Auth, always use HTTPS for TLS/SSL encryption, store passwords hashed using bcrypt or argon2, implement rate limiting, and add request signing.

Best Practices

Use Environment Variables
Never hardcode credentials in source code. Store them in environment variables instead.

Implement Rate Limiting
Limit the number of requests per IP address or user within a time window to prevent brute force attacks. Return 429 (Too Many Requests) when limits are exceeded.

Add Request Logging
Log all authentication attempts and API access for security monitoring and audit purposes.

Input Validation
Always validate and sanitize user inputs to prevent injection attacks and ensure data integrity.

CORS (Cross-Origin Resource Sharing)

The API includes basic CORS headers that allow all origins. In production environments, you should specify allowed domains instead of using the wildcard.

```
Access-Control-Allow-Origin: * Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS Access-Control-Allow-Headers: Content-Type, Authorization
```

Note: The wildcard (*) allows all origins. In production, specify allowed domains like Access-Control-Allow-Origin: https://yourdomain.com

API Versioning

This API does not currently implement versioning. For production environments, consider implementing version control through URL versioning or header versioning.

URL Versioning Example:

```
http://localhost:8000/v1/transactions http://localhost:8000/v2/transactions
```

Header Versioning Example:

```
GET /transactions Accept: application/vnd.api.v1+json
```

Last Updated: February 2, 2026