readme.md for @apiclient.xyz/bunq

npm version TypeScript License: MIT

A powerful, type-safe TypeScript/JavaScript client for the bunq API with full feature coverage

Table of Contents

Features

Core Banking Operations

- □□ **Complete Account Management** - Access all account types (personal, business, joint)
- □□ **Advanced Payment Processing** - Single payments, batch payments, scheduled payments
- □□ **Transaction History** - Full transaction access with filtering and pagination
- □□ **Payment Requests** - Send and manage payment requests with bunq.me integration
- □□ **Draft Payments** - Create payments requiring approval

Advanced Features

- □□ **Automatic Session Management** - Handles token refresh and session renewal
- □□ **Full Security Implementation** - Request signing and response verification
- □□ **Webhook Support** - Real-time notifications with signature verification
- □□ **Card Management** - Full card control (activation, limits, blocking)
- □□ **File Attachments** - Upload and attach files to payments
- □□ **Statement Exports** - Export statements in multiple formats (PDF, CSV, MT940)
- □□ **OAuth Support** - Third-party app integration
- □□ **Sandbox Environment** - Full testing support

Developer Experience

- □□ **Full TypeScript Support** - Complete type definitions for all API responses
- □□ **Builder Pattern APIs** - Intuitive payment and request builders
- ⚡ **Promise-based** - Modern async/await support throughout
- □□ **Type Safety** - Compile-time type checking for all operations
- □□ **Comprehensive Documentation** - Detailed examples for every feature
- □□ **Robust HTTP Client** - Built-in retry logic and rate limit handling

Stateless Architecture (v4.0.0+)

Starting from version 4.0.0, this library is completely stateless. Session management is now entirely controlled by the consumer:

Key Changes

- **No File Persistence** - The library no longer saves any state to disk
- **Session Data Export** - Full session data is returned for you to persist
- **Session Data Import** - Initialize with previously saved session data
- **Explicit Session Management** - You control when and how sessions are stored

Benefits

- **Full Control** - Store sessions in your preferred storage (database, Redis, etc.)
- **Better for Microservices** - No shared state between instances
- **Improved Testing** - Predictable behavior with no hidden state
- **Enhanced Security** - You control where sensitive data is stored

Migration from v3.x

If you're upgrading from v3.x, you'll need to handle session persistence yourself. See the Stateless Session Management section for examples.

Installation

```
npm install @apiclient.xyz/bunq
```

```
yarn add @apiclient.xyz/bunq
```

```
pnpm add @apiclient.xyz/bunq
```

Quick Start

```
import { BunqAccount } from '@apiclient.xyz/bunq';

// Initialize the client
const bunq = new BunqAccount({
  apiKey: 'your-api-key',
```

```
  deviceName: 'My App',
  environment: 'PRODUCTION' // or 'SANDBOX' for testing
});

// Initialize connection and get session data
const sessionData = await bunq.init();

// IMPORTANT: Save the session data for reuse
await saveSessionToDatabase(sessionData);

// Get your accounts
const { accounts, sessionData: updatedSession } = await bunq.getAccounts();
console.log(`Found ${accounts.length} accounts`);

// If session was refreshed, save the updated data
if (updatedSession) {
  await saveSessionToDatabase(updatedSession);
}

// Get recent transactions
const transactions = await accounts[0].getTransactions();
transactions.forEach(tx => {
  console.log(`${tx.created}: ${tx.amount.value} ${tx.amount.currency} - ${tx.description}`);
});

// Always cleanup when done
await bunq.stop();
```

Core Examples

Account Management

```
// Get all accounts with details
const accounts = await bunq.getAccounts();

for (const account of accounts) {
  console.log(`Account: ${account.description}`);
```

```
  console.log(`Balance: ${account.balance.value} ${account.balance.currency}`);
  console.log(`IBAN: ${account.iban}`);

  // Get account-specific transactions
  const transactions = await account.getTransactions({
    count: 50,  // Last 50 transactions
    newer_id: false,
    older_id: false
  });
}

// Create a new monetary account (business accounts only)
const newAccount = await BunqMonetaryAccount.create(bunq, {
  currency: 'EUR',
  description: 'Savings Account',
  dailyLimit: '1000.00',
  overdraftLimit: '0.00'
});
```

Making Payments

Simple Payment

```
// Using the payment builder pattern
const payment = await BunqPayment.builder(bunq, account)
  .amount('25.00', 'EUR')
  .toIban('NL91ABNA0417164300', 'John Doe')
  .description('Birthday gift')
  .create();

console.log(`Payment created with ID: ${payment.id}`);
```

Payment with Custom Request ID (Idempotency)

```
// Prevent duplicate payments with custom request IDs
const payment = await BunqPayment.builder(bunq, account)
  .amount('100.00', 'EUR')
  .toIban('NL91ABNA0417164300', 'Supplier B.V.')
  .description('Invoice #12345')
```

```
  .customRequestId('invoice-12345-payment')  // Prevents duplicate payments
  .create();
```

Batch Payments

```
const batch = new BunqPaymentBatch(bunq);

// Create multiple payments in one API call
const batchId = await batch.create(account, [
  {
    amount: { value: '10.00', currency: 'EUR' },
    counterparty_alias: {
      type: 'IBAN',
      value: 'NL91ABNA0417164300',
      name: 'Employee 1'
    },
    description: 'Salary payment'
  },
  {
    amount: { value: '20.00', currency: 'EUR' },
    counterparty_alias: {
      type: 'EMAIL',
      value: 'freelancer@example.com',
      name: 'Freelancer'
    },
    description: 'Project payment'
  }
]);

// Check batch status
const batchDetails = await batch.get(account, batchId);
console.log(`Batch status: ${batchDetails.status}`);
console.log(`Total amount: ${batchDetails.total_amount.value}`);
```

Scheduled & Recurring Payments

```
const scheduler = new BunqSchedulePayment(bunq);

// One-time scheduled payment
```

```
const tomorrow = new Date();
tomorrow.setDate(tomorrow.getDate() + 1);

const scheduledId = await BunqSchedulePayment.builder(bunq, account)
  .amount('50.00', 'EUR')
  .toIban('NL91ABNA0417164300', 'Landlord')
  .description('Rent payment')
  .scheduleOnce(tomorrow.toISOString())
  .create();

// Recurring monthly payment
const recurringId = await BunqSchedulePayment.builder(bunq, account)
  .amount('9.99', 'EUR')
  .toIban('NL91ABNA0417164300', 'Netflix B.V.')
  .description('Monthly subscription')
  .scheduleMonthly('2024-01-01T10:00:00Z', '2024-12-31T10:00:00Z')
  .create();

// List all scheduled payments
const schedules = await scheduler.list(account);

// Cancel a scheduled payment
await scheduler.delete(account, scheduledId);
```

Payment Requests

```
// Create a payment request
const request = await BunqRequestInquiry.builder(bunq, account)
  .amount('25.00', 'EUR')
  .fromEmail('friend@example.com', 'My Friend')
  .description('Lunch money')
  .allowBunqme()  // Generate bunq.me link
  .minimumAge(18)
  .create();

console.log(`Share this link: ${request.bunqmeShareUrl}`);

// List pending requests
```

```
const requests = await BunqRequestInquiry.list(bunq, account.id);
const pending = requests.filter(r => r.status === 'PENDING');

// Cancel a request
await request.update(requestId, { status: 'CANCELLED' });
```

Draft Payments (Requires Approval)

```
const draft = new BunqDraftPayment(bunq, account);

// Create a draft with multiple payments
const draftId = await draft.create({
  numberOfRequiredAccepts: 2,  // Requires 2 approvals
  entries: [
    {
      amount: { value: '1000.00', currency: 'EUR' },
      counterparty_alias: {
        type: 'IBAN',
        value: 'NL91ABNA0417164300',
        name: 'Supplier A'
      },
      description: 'Invoice payment'
    },
    {
      amount: { value: '2000.00', currency: 'EUR' },
      counterparty_alias: {
        type: 'IBAN',
        value: 'NL91ABNA0417164300',
        name: 'Supplier B'
      },
      description: 'Equipment purchase'
    }
  ]
});

// Approve the draft
await draft.accept();
```

```
// Or reject it
await draft.reject('Budget exceeded');
```

Card Management

```
// List all cards
const cards = await BunqCard.list(bunq);

// Get card details
for (const card of cards) {
  console.log(`Card: ${card.name_on_card}`);
  console.log(`Status: ${card.status}`);
  console.log(`Type: ${card.type}`)
  console.log(`Expiry: ${card.expiry_date}`);

  // Get card limits
  const limits = card.limit;
  console.log(`Daily limit: ${limits.daily_spent}`);
}

// Note: Card management methods like activation, PIN updates, and ordering
// new cards should be performed through the bunq app or API directly.
```

Webhooks

```
// Setup webhook server
const webhookServer = new BunqWebhookServer(bunq, {
  port: 3000,
  publicUrl: 'https://myapp.com/webhooks'
});

// Register event handlers
webhookServer.getHandler().onPayment((payment) => {
  console.log(`New payment: ${payment.amount.value} ${payment.amount.currency}`);
  console.log(`From: ${payment.counterparty_alias.display_name}`);
  console.log(`Description: ${payment.description}`);
```

```
  // Your business logic here
  updateDatabase(payment);
  sendNotification(payment);
});

webhookServer.getHandler().onRequest((request) => {
  console.log(`New payment request: ${request.amount_inquired.value}`);
  console.log(`From: ${request.user_alias_created.display_name}`);
});

webhookServer.getHandler().onCard((card) => {
  if (card.status === 'BLOCKED') {
    console.log(`Card blocked: ${card.name_on_card}`);
    alertSecurityTeam(card);
  }
});

// Start server and register with bunq
await webhookServer.start();
await webhookServer.register();

// Manual webhook management
const webhook = new BunqWebhook(bunq, account);

// Create webhook for specific URL
const webhookId = await webhook.create(account, 'https://myapp.com/bunq-webhook');

// List all webhooks
const webhooks = await webhook.list(account);

// Delete webhook
await webhook.delete(account, webhookId);
```

File Attachments

```
const attachment = new BunqAttachment(bunq);

// Upload a file
```

```
const attachmentUuid = await attachment.uploadFile(
  '/path/to/invoice.pdf',
  'Invoice #12345'
);

// Attach to payment
const payment = await BunqPayment.builder(bunq, account)
  .amount('150.00', 'EUR')
  .toIban('NL91ABNA0417164300', 'Accountant')
  .description('Services rendered')
  .attachments([attachmentUuid])
  .create();

// Upload from buffer
const buffer = await generateReport();
const uuid = await attachment.uploadBuffer(
  buffer,
  'report.pdf',
  'application/pdf',
  'Monthly Report'
);

// Get attachment content
const content = await attachment.getContent(attachmentUuid);
await fs.writeFile('downloaded.pdf', content);
```

Export Statements

```
// Export last month as PDF
await new ExportBuilder(bunq, account)
  .asPdf()
  .lastMonth()
  .downloadTo('/path/to/statement.pdf');

// Export date range as CSV
await new ExportBuilder(bunq, account)
  .asCsv()
  .dateRange('2024-01-01', '2024-03-31')
```

```
  .regionalFormat('EUROPEAN')
  .downloadTo('/path/to/transactions.csv');

// Export as MT940 for accounting software
await new ExportBuilder(bunq, account)
  .asMt940()
  .lastDays(90)  // Last 90 days
  .downloadTo('/path/to/statement.sta');

// Export last 30 days with attachments
await new ExportBuilder(bunq, account)
  .asPdf()
  .lastDays(30)
  .includeAttachments(true)
  .downloadTo('/path/to/statement-with-attachments.pdf');

// Get statement as Buffer (no file saving)
const buffer = await new ExportBuilder(bunq, account)
  .asPdf()
  .lastMonth()
  .download();
// Use the buffer directly, e.g., send as email attachment
await emailService.sendWithAttachment(buffer, 'statement.pdf');

// Get statement as ArrayBuffer for web APIs
const arrayBuffer = await new ExportBuilder(bunq, account)
  .asCsv()
  .lastDays(30)
  .downloadAsArrayBuffer();
// Use with web APIs like Blob
const blob = new Blob([arrayBuffer], { type: 'text/csv' });

// Using account's getAccountStatement method for easy month selection
const statement1 = account.getAccountStatement({
  monthlyIndexedFrom1: 1,  // Last month (1 = last month, 2 = two months ago, etc.)
  includeTransactionAttachments: true
});
await statement1.asPdf().downloadTo('/path/to/last-month.pdf');
```

```
// Or using 0-based indexing
const statement2 = account.getAccountStatement({
  monthlyIndexedFrom0: 0,  // Current month (0 = current, 1 = last month, etc.)
  includeTransactionAttachments: false
});
await statement2.asCsv().downloadTo('/path/to/current-month.csv');

// Or specify exact date range
const statement3 = account.getAccountStatement({
  from: new Date('2024-01-01'),
  to: new Date('2024-03-31'),
  includeTransactionAttachments: true
});
await statement3.asMt940().downloadTo('/path/to/q1-statement.sta');
```

Stateless Session Management

```
// Initial session creation
const bunq = new BunqAccount({
  apiKey: 'your-api-key',
  deviceName: 'My App',
  environment: 'PRODUCTION'
});

// Initialize and receive session data
const sessionData = await bunq.init();

// Save to your preferred storage
await saveToDatabase({
  userId: 'user123',
  sessionData: sessionData,
  createdAt: new Date()
});

// Reusing existing session
const savedData = await loadFromDatabase('user123');
const bunq2 = new BunqAccount({
  apiKey: 'your-api-key',
```

```
  deviceName: 'My App',
  environment: 'PRODUCTION'
});

// Initialize with saved session
await bunq2.initWithSession(savedData.sessionData);

// Check if session is valid
if (!bunq2.isSessionValid()) {
  // Session expired, create new one
  const newSession = await bunq2.init();
  await saveToDatabase({ userId: 'user123', sessionData: newSession });
}

// Making API calls with automatic refresh
const { accounts, sessionData: refreshedSession } = await bunq2.getAccounts();

// Always save refreshed session data
if (refreshedSession) {
  await saveToDatabase({
    userId: 'user123',
    sessionData: refreshedSession,
    updatedAt: new Date()
  });
}

// Get current session data at any time
const currentSession = bunq2.getSessionData();
```

User Management

```
// Get user information
const user = await bunq.getUser();
const userInfo = await user.getInfo();

// Determine user type
if (userInfo.UserPerson) {
  console.log(`Personal account: ${userInfo.UserPerson.display_name}`);
```

```
} else if (userInfo.UserCompany) {
  console.log(`Business account: ${userInfo.UserCompany.name}`);
}

// Update user settings
await user.update({
  dailyLimitWithoutConfirmationLogin: '100.00',
  notificationFilters: [
    { category: 'PAYMENT', notificationDeliveryMethod: 'PUSH' }
  ]
});
```

Advanced Usage

Custom Request Headers

```
// Use custom request IDs for idempotency
const payment = await BunqPayment.builder(bunq, account)
  .amount('100.00', 'EUR')
  .toIban('NL91ABNA0417164300', 'Recipient')
  .description('Invoice payment')
  .customRequestId('unique-request-id-123')  // Prevents duplicate payments
  .create();

// The same request ID will return the original payment without creating a duplicate
```

OAuth Token Support

```
// Using OAuth access token instead of API key
const bunq = new BunqAccount({
  apiKey: 'your-oauth-access-token',  // OAuth token from bunq OAuth flow
  deviceName: 'OAuth App',
  environment: 'PRODUCTION',
  isOAuthToken: true  // Important for OAuth-specific handling
});
```

```
try {
  // Try normal initialization
  const sessionData = await bunq.init();
  await saveOAuthSession(sessionData);
} catch (error) {
  // OAuth token may already have installation/device
  if (error.message.includes('already has a user session')) {
    // Load existing installation data if available
    const existingInstallation = await loadOAuthInstallation();

    // Initialize with existing installation
    const sessionData = await bunq.initOAuthWithExistingInstallation(existingInstallation);
    await saveOAuthSession(sessionData);
  } else {
    throw error;
  }
}

// Use the OAuth-initialized account normally
const { accounts, sessionData } = await bunq.getAccounts();

// OAuth tokens work like regular API keys:
// 1. They go through installation → device → session creation
// 2. The OAuth token is used as the 'secret' during authentication
// 3. A session token is created and used for all API calls
```

Error Handling

```
import { BunqApiError } from '@apiclient.xyz/bunq';

try {
  await payment.create();
} catch (error) {
  if (error instanceof BunqApiError) {
    // Handle API errors
    console.error('API Error:', error.errors);
    error.errors.forEach(e => {
```

```
      console.error(`- ${e.error_description}`);
    });
  } else if (error.response?.status === 401) {
    // Handle authentication errors
    console.error('Authentication failed:', error.message);
    await bunq.init();  // Re-initialize session
  } else {
    // Handle other errors
    console.error('Unexpected error:', error);
  }
}
```

Automatic Rate Limit Handling

The library automatically handles rate limiting (HTTP 429 responses) with intelligent exponential backoff:

```
// No special handling needed - rate limits are handled automatically
const payments = await Promise.all([
  payment1.create(),
  payment2.create(),
  payment3.create(),
  // ... many more payments
]);

// The HTTP client will automatically:
// 1. Detect 429 responses
// 2. Wait for the time specified in Retry-After header
// 3. Use exponential backoff if no Retry-After is provided
// 4. Retry the request automatically (up to 3 times)
```

Pagination

```
// Paginate through all transactions
async function* getAllTransactions(account: BunqMonetaryAccount) {
  let olderId: number | false = false;

  while (true) {
```

```
    const transactions = await account.getTransactions({
      count: 200,
      older_id: olderId
    });

    if (transactions.length === 0) break;

    yield* transactions;
    olderId = transactions[transactions.length - 1].id;
  }
}

// Usage
for await (const transaction of getAllTransactions(account)) {
  console.log(`${transaction.created}: ${transaction.description}`);
}
```

Sandbox Testing

```
// Create sandbox environment
const sandboxBunq = new BunqAccount({
  apiKey: '',  // Will be generated
  deviceName: 'My Test App',
  environment: 'SANDBOX'
});

// Create sandbox user with €1000 balance
const apiKey = await sandboxBunq.createSandboxUser();
console.log('Sandbox API key:', apiKey);

// Re-initialize with the generated key
const bunq = new BunqAccount({
  apiKey: apiKey,
  deviceName: 'My Test App',
  environment: 'SANDBOX'
});
await bunq.init();
```

```
// The sandbox environment provides €1000 initial balance for testing
// Additional sandbox-specific features can be accessed through the bunq API directly
```

Security Best Practices

1. **API Key Storage**: Never commit API keys to version control

```
const bunq = new BunqAccount({
  apiKey: process.env.BUNQ_API_KEY,
  deviceName: 'Production App',
  environment: 'PRODUCTION'
});
```

2. **IP Whitelisting**: Restrict API access to specific IPs

```
const bunq = new BunqAccount({
  apiKey: process.env.BUNQ_API_KEY,
  permittedIps: ['1.2.3.4', '5.6.7.8']
});
```

3. **Webhook Verification**: Always verify webhook signatures

```
app.post('/webhook', (req, res) => {
  const signature = req.headers['x-bunq-client-signature'];
  const isValid = bunq.verifyWebhookSignature(req.body, signature);

  if (!isValid) {
    return res.status(401).send('Invalid signature');
  }

  // Process webhook...
});
```

Migration Guide

From @bunq-community/bunq-js-client

```
// Old
import BunqJSClient from '@bunq-community/bunq-js-client';
const bunqJSClient = new BunqJSClient();

// New
import { BunqAccount } from '@apiclient.xyz/bunq';
const bunq = new BunqAccount({
  apiKey: 'your-api-key',
  deviceName: 'My App'
});

// Old
await bunqJSClient.install();
await bunqJSClient.registerDevice();
await bunqJSClient.registerSession();

// New - all handled in one call
await bunq.init();
```

Migration Guide from v3.x to v4.0.0

Version 4.0.0 introduces a breaking change: the library is now completely stateless. Here's how to migrate:

Before (v3.x)

```
// Session was automatically saved to .nogit/bunqproduction.json
const bunq = new BunqAccount({ apiKey, deviceName, environment });
await bunq.init(); // Session saved to disk automatically
const accounts = await bunq.getAccounts(); // Returns accounts directly
```

After (v4.0.0)

```
// You must handle session persistence yourself
const bunq = new BunqAccount({ apiKey, deviceName, environment });
const sessionData = await bunq.init(); // Returns session data
await myDatabase.save('session', sessionData); // You save it

// API calls now return both data and potentially refreshed session
const { accounts, sessionData: newSession } = await bunq.getAccounts();
if (newSession) {
  await myDatabase.save('session', newSession); // Save refreshed session
}
```

Key Changes

1. **No automatic file persistence** - Remove any dependency on `.nogit/` files
2. `init()` **returns session data** - You must save this data yourself
3. **API methods return objects** - Methods like `getAccounts()` now return `{ accounts, sessionData? }`
4. **Session reuse requires explicit loading** - Use `initWithSession(savedData)`
5. **OAuth handling is explicit** - Use `initOAuthWithExistingInstallation()` for OAuth tokens with existing installations

Session Storage Example

```
// Simple file-based storage (similar to v3.x behavior)
import { promises as fs } from 'fs';

async function saveSession(data: ISessionData) {
  await fs.writeFile('./my-session.json', JSON.stringify(data));
}

async function loadSession(): Promise<ISessionData | null> {
  try {
    const data = await fs.readFile('./my-session.json', 'utf-8');
    return JSON.parse(data);
  } catch {
    return null;
  }
}
```

```
// Database storage example
async function saveSessionToDB(userId: string, data: ISessionData) {
  await db.collection('bunq_sessions').updateOne(
    { userId },
    { $set: { sessionData: data, updatedAt: new Date() } },
    { upsert: true }
  );
}
```

Testing

The library includes comprehensive test coverage:

```
# Run all tests
npm test

# Run specific test suites
npm run test:basic      # Core functionality
npm run test:payments   # Payment features
npm run test:webhooks   # Webhook functionality
npm run test:session    # Session management
npm run test:errors     # Error handling
npm run test:advanced   # Advanced features
```

Requirements

- Node.js 14.x or higher
- TypeScript 4.5 or higher (for TypeScript users)

License and Legal Information

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH and are not included within the scope of the MIT license granted herein. Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines, and any usage must be approved in writing by Task Venture Capital GmbH.

Company Information

Task Venture Capital GmbH
Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #7
Created 2026-03-28 10:48:12 UTC by foss.global Team
Updated 2026-03-28 12:13:22 UTC by foss.global Team