REST API Specifications

Base URL : https://backtrade.damien-reichhart.fr/api
Format : JSON
Authentication : JWT (JSON Web Token)

Required Headers

For all requests :

```
Content-Type: application/json
```

For protected routes :

```
Authorization: Bearer {token}
```

For requests with file upload :

```
Content-Type: multipart/form-data
Authorization: Bearer {token}
```

CORS

The server must allow requests from the react frontend running on https://backtrade.damien-reichhart.fr.

Required CORS configuration :

- Origin : https://backtrade.damien-reichhart.fr
- Methods : GET, POST, PUT, DELETE, PATCH
- Headers : Origin, X-Requested-With, Content-Type, Accept, Authorization

Authentication

The API uses JWT tokens to secure endpoints.

JWT Token Format

The token contains the following information :

```
{
    "sub": {
        "id": 1,
        "email": "admin@backtrade.com",
        "role": "ADMIN",
        "is_banned": false,
        "stripe_customer_id": "cus_admin123",
        "created_at": "2024-01-01T00:00:00.000Z",
        "updated_at": "2024-01-15T10:30:00.000Z"
    },
    "iat": 123456,
    "exp": 123456
}
```


Validity duration : 1 hour

Token Usage

The token must be included in the Authorization header with the Bearer prefix :

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Token Renewal

The token is not automatically renewable or Refresh token

HTTP Codes

Code	Status	When to use
200	OK	Request successful
201	Created	Resource created successfully
400	Bad Request	Malformed request or invalid data
401	Unauthorized	Authentication required or invalid token
403	Forbidden	Access denied (user is not the resource owner)
404	Not Found	Resource not found
409	Conflict	Conflict (e.g., email already used)
500	Internal Server Error	Server error

Error Format

```
{
    "error": {
        "message": "Error description",
        "code": "error code"
    }
}
```

Schemas

Name	Content	Description
PaginationQuery	`{ page?: number, limit?: number, sort?: string, order?: 'asc' | 'desc' }`	Common pagination parameters
SearchQuery	`{ q?: string } & PaginationQuery`	Common search parameters
DateRangeQuery	`{ ts_gte?: string, ts_lte?: string } & PaginationQuery`	Common date range filter parameters
PositionQuery	`{ session_id?: number, status?: PositionStatus } & PaginationQuery`	Position query parameters with optional session and status filters
PublicUser	`{ id: number, email: string, role: Role, is_banned: boolean, stripe_customer_id?: string, created_at: string, updated_at: string }`	Public user data (no password hash)
Role	`'ADMIN' | 'USER'`	User role
LoginRequest	`{ email: string, password: string }`	Request body for login
RegisterRequest	`{ email: string, password: string, confirmPassword: string }`	Request body for registration
RefreshTokenRequest	`{ refreshToken: string }`	Request body for token refresh
ChangePasswordRequest	`{ currentPassword: string, newPassword: string }`	Request body for changing password (newPassword min 8 chars)

Name	Content	Description
ForgotPasswordRequest	`{ email: string }`	Request body for password reset email
ResetPasswordRequest	`{ email: string, code: string, newPassword: string }`	Request body for password reset with email, code, and new password (min 8 chars)
UpdateUserRequest	`{ email?: string, role?: Role, is_banned?: boolean }`	Request body for updating user details (all fields optional)
AuthResponse	`{ user: PublicUser, accessToken: string, refreshToken: string }`	Response for authentication endpoints
Session	`{ id: number, user_id: number, instrument_id: number, name?: string, session_status: SessionStatus, speed: Speed, start_time: string, current_time: string, end_time?: string, initial_balance: number, leverage: Leverage, spread_pts: number, slippage_pts: number, commission_per_fill: number, created_at: string, updated_at: string }`	Trading session entity
CreateSessionRequest	`{ instrument_id: number, name?: string, speed: Speed, start_time: string, current_time: string, end_time?: string, initial_balance: number, leverage: Leverage, spread_pts: number, slippage_pts: number, commission_per_fill: number, session_status?: SessionStatus }`	Payload to create a session (user_id set automatically, current_time must equal start_time)
UpdateSessionRequest	`{ name?: string, session_status?: SessionStatus, speed?: Speed, current_time?: string, end_time?: string }`	Payload to update a session

Name	Content	Description
Plan	`{ id: number, code: string, stripe_product_id: string, stripe_price_id: string, currency: string, price: number, max_active_sessions: number }`	Subscription plan entity
CreatePlanRequest	`{ code: string, stripe_product_id: string, stripe_price_id: string, currency: string, price: number, max_active_sessions: number }`	Payload to create a plan
UpdatePlanRequest	`{ code?: string, stripe_product_id?: string, stripe_price_id?: string, currency?: string, price?: number, max_active_sessions?: number }`	Payload to update a plan
Subscription	`{ id: number, user_id: number, plan_id: number, stripe_subscription_id: string, status: SubscriptionStatus, current_period_start: string, current_period_end: string, cancel_at_period_end: boolean }`	User subscription entity
CreateSubscriptionRequest	`{ user_id: number, plan_id: number, stripe_subscription_id: string, current_period_start: string, current_period_end: string, status?: SubscriptionStatus, cancel_at_period_end: boolean }`	Payload to create a subscription
UpdateSubscriptionRequest	`{ status?: SubscriptionStatus, cancel_at_period_end?: boolean }`	Payload to update a subscription
Transaction	`{ id: number, session_id?: number, transaction_type: TransactionType, amount: number, balance_after: number, created_at: string, updated_at: string }`	Financial transaction entity
CreateTransactionRequest	`{ session_id?: number, transaction_type: TransactionType, amount: number, balance_after: number }`	Payload to create a transaction

Name	Content	Description
Position	`{ id: number, session_id: number, position_status: PositionStatus, side: Side, quantity_lots: number, tp_price?: number, sl_price?: number, entry_price: number, exit_price?: number, opened_at: string, closed_at?: string, realized_pnl?: number, unrealized_pnl?: number, commission_cost?: number, slippage_cost?: number, spread_cost?: number, created_at: string, updated_at: string }`	Trading position entity
CreatePositionRequest	`{ session_id: number, side: Side, entry_price: number, quantity_lots: number, tp_price?: number, sl_price?: number, position_status: PositionStatus, opened_at: string }`	Payload to create a position
UpdatePositionRequest	`{ position_status?: PositionStatus, exit_price?: number, closed_at?: string, realized_pnl?: number, commission_cost?: number, slippage_cost?: number, spread_cost?: number, tp_price?: number, sl_price?: number }`	Payload to update a position
Dataset	`{ id: number, instrument_id: number, timeframe: Timeframe, uploaded_at?: string, records_count?: number, file_name?: string, start_time?: string, end_time?: string, created_at: string, updated_at: string }`	Dataset entity
CreateDatasetRequest	`{ instrument_id: number, timeframe: Timeframe }`	Payload to create a dataset
UpdateDatasetRequest	`{ instrument_id: number, timeframe: Timeframe }`	Payload to update a dataset
Instrument	`{ id: number, symbol: string, display_name: string, pip_size: number, created_at: string, updated_at: string }`	Trading instrument entity
CreateInstrumentRequest	`{ symbol: string, display_name: string, pip_size: number }`	Payload to create an instrument
UpdateInstrumentRequest	`{ display_name?: string, pip_size?: number }`	Payload to update an instrument

SessionAnalyticsResponse	`{ summary: AnalyticsSummary, equity_curve: EquityCurvePoint[], breakdowns: AnalyticsBreakdowns, costs: AnalyticsCosts, top_winners: Position[], worst_losers: Position[], daily_pnl: DailyPnL[] }`	Detailed analytics response for a session
AnalyticsSummary	`{ total_trades: number, expectancy: number, commission_paid: number, return_percentage: number, win_rate: number, profit_factor: number, start_balance: number, ending_equity: number, net_pnl: number, sharpe_ratio: number, sortino_ratio: number, max_drawdown: number, win_streak: number, lose_streak: number }`	Performance summary metrics including win/loss streaks calculated from all closed positions in chronological order
SessionSkipResponse	`{ session: Session, candle: Candle }`	Response for skipping to the next candle, includes updated session and new candle data

API Routes

Verb	Endpoint	Query Params	Request body	Response type	Status codes	Description
POST	`/auth/login`	-	`LoginRequest`	`AuthResponse`	200, 400, 401	Authenticate user
POST	`/auth/register`	-	`RegisterRequest`	`AuthResponse`	201, 400, 409	Register new user
POST	`/auth/refresh-token`	-	`RefreshTokenRequest`	`AuthResponse`	200, 400, 401	Refresh access token
GET	`/auth/me`	-	-	`PublicUser`	200, 401	Get current authenticated user
POST	`/auth/users/requester/password`	-	`ForgotPasswordRequest`	-	200, 400	Request password reset email (sends 6-digit code, expires in 15 min)
POST	`/auth/users/resetter/password`	-	`ResetPasswordRequest`	-	200, 400	Reset password with email, code, and new password
GET	`/health`	-	-	`{ status: string }`	200	Health check endpoint
GET	`/users`	`SearchQuery`	-	`PublicUser[]`	200, 401, 403	List all users (Admin only)
GET	`/users/:id`	-	-	`PublicUser`	200, 401, 403, 404	Get user by ID
GET	`/users/:id/subscriptions`	`DateRangeQuery`	-	`Subscription[]`	200, 401, 404	List subscriptions by user
PATCH	`/users/:id/password`	-	`ChangePasswordRequest`	-	200, 400, 401	Change user password
PATCH	`/users/:id`	-	`UpdateUserRequest`	`PublicUser`	200, 400, 401, 403, 404, 409	Update user details
DELETE	`/users/:id`	-	-	-	204, 401, 403, 404	Delete user
GET	`/sessions`	`SearchQuery`	-	`Session[]`	200, 401	List user sessions
GET	`/sessions/:id`	-	-	`Session`	200, 401, 404	Get session by ID
POST	`/sessions`	-	`CreateSessionRequest`	`Session`	201, 400, 401, 403	Create new session
PATCH	`/sessions/:id`	-	`UpdateSessionRequest`	`Session`	200, 400, 401, 404	Update session
DELETE	`/sessions/:id`	-	-	-	204, 401, 404	Delete session
GET	`/sessions/:id/candles`	`timeframe`	-	`Candle[]`	200, 401, 403, 404	List last 2000 candles for a session in a specified timeframe
GET	`/sessions/:id/info`	-	-	`SessionInfo`	200, 401, 403, 404	Get session information (current balance, equity, margin, etc.)
GET	`/sessions/:id/analytics`	-	-	`SessionAnalyticsResponse`	200, 401, 403, 404	Get detailed session analytics
PATCH	`/sessions/:id/skip`	`timeframe`	-	`SessionSkipResponse`	200, 400, 401, 403, 404	Skip to the next candle for a session (advances current_time, processes TP/SL checks)

Verb	Endpoint	Query Params	Request body	Response type	Status codes	Description
GET	`/sessions/:id/positions`	`PositionQuery`	-	`Position[]`	200, 401, 404	List positions by session
GET	`/sessions/:id/transactions`	`PaginationQuery`	-	`Transaction[]`	200, 401, 404	List transactions by session
PUT	`/sessions/:id`	-	`UpdateSessionRequest`	`Session`	200, 400, 401, 404	Update session (alternative to PATCH)
GET	`/plans`	`SearchQuery`	-	`Plan[]`	200, 401	List subscription plans (Auth required)
GET	`/plans/:id`	-	-	`Plan`	200, 401, 404	Get plan by ID (Auth required)
POST	`/plans`	-	`CreatePlanRequest`	`Plan`	201, 400, 401, 403	Create plan (Admin)
PATCH	`/plans/:id`	-	`UpdatePlanRequest`	`Plan`	200, 400, 401, 403, 404	Update plan (Admin)
DELETE	`/plans/:id`	-	-	-	204, 401, 403, 404	Delete plan (Admin)
GET	`/subscriptions`	`DateRangeQuery`	-	`Subscription[]`	200, 401	List user subscriptions
GET	`/subscriptions/:id`	-	-	`Subscription`	200, 401, 404	Get subscription by ID
POST	`/subscriptions`	-	`CreateSubscriptionRequest`	`Subscription`	201, 400, 401, 403	Create subscription (Admin only)
PATCH	`/subscriptions/:id`	-	`UpdateSubscriptionRequest`	`Subscription`	200, 400, 401, 403, 404	Update subscription (Admin only)
DELETE	`/subscriptions/:id`	-	-	-	204, 401, 403, 404	Cancel subscription (Admin only)
GET	`/transactions`	`DateRangeQuery`	-	`Transaction[]`	200, 401	List user transactions
GET	`/transactions/:id`	-	-	`Transaction`	200, 401, 404	Get transaction by ID
POST	`/transactions`	-	`CreateTransactionRequest`	`Transaction`	201, 400, 401	Create transaction
GET	`/positions`	`PositionQuery` (session_id?)	-	`Position[]`	200, 401	List all positions (optionally filtered by session_id)
GET	`/positions/:id`	-	-	`Position`	200, 401, 404	Get position by ID
POST	`/positions`	-	`CreatePositionRequest`	`Position`	201, 400, 401	Open new position
PUT	`/positions/:id`	-	`UpdatePositionRequest`	`Position`	200, 400, 401, 404	Update position (alternative to PATCH, use to close position)
PATCH	`/positions/:id`	-	`UpdatePositionRequest`	`Position`	200, 400, 401, 404	Update position (use to close position)
PATCH	`/sessions/:id/positions`	`closeAll=true`	-	`{ closed: number, failed: number, total: number }`	200, 400, 401, 404	Close all positions for session
DELETE	`/positions/:id`	-	-	-	204, 401, 404	Delete position
GET	`/datasets`	`DateRangeQuery`	-	`Dataset[]`	200, 401, 403	List datasets (Admin only)
GET	`/datasets/:id`	-	-	`Dataset`	200, 401, 403, 404	Get dataset by ID (Admin only)
POST	`/datasets`	-	`CreateDatasetRequest`	`Dataset`	201, 400, 401, 403	Create dataset (Admin only)
POST	`/datasets/:id/file`	-	`multipart/form-data`	`Dataset`	200, 400, 401, 403, 404	Upload dataset file (Admin only)
PUT	`/datasets/:id`	-	`UpdateDatasetRequest`	`Dataset`	200, 400, 401, 403, 404	Update dataset (Admin only, alternative to PATCH)
PATCH	`/datasets/:id`	-	`UpdateDatasetRequest`	`Dataset`	200, 400, 401, 403, 404	Update dataset (Admin only)
DELETE	`/datasets/:id`	-	-	-	204, 401, 403, 404	Delete dataset (Admin only)
GET	`/instruments`	`SearchQuery`	-	`Instrument[]`	200, 401	List instruments (Auth required)

Verb	Endpoint	Query Params	Request body	Response type	Status codes	Description
GET	`/instruments/:id`	-	-	`Instrument`	200, 401, 404	Get instrument by ID (Auth required)
POST	`/instruments`	-	`CreateInstrumentRequest`	`Instrument`	201, 400, 401, 403	Create instrument (Admin only)
PUT	`/instruments/:id`	-	`UpdateInstrumentRequest`	`Instrument`	200, 400, 401, 403, 404	Update instrument (Admin only, alternative to PATCH)
PATCH	`/instruments/:id`	-	`UpdateInstrumentRequest`	`Instrument`	200, 400, 401, 403, 404	Update instrument (Admin only)
DELETE	`/instruments/:id`	-	-	-	204, 401, 403, 404	Delete instrument (Admin only)
POST	`/stripe/checkout`	-	`{ planId: number }`	`{ sessionId: string, url: string }`	200, 400, 401	Create Stripe checkout session
POST	`/stripe/portal`	-	-	`{ url: string }`	200, 401	Create Stripe customer portal session
GET	`/stripe/checkout/:sessionId`	-	-	`{ status: string, subscriptionId: string | null, customerId: string | null }`	200, 400, 401	Get checkout session status
POST	`/stripe/webhook`	-	`Raw JSON (Stripe event)`	-	200	Stripe webhook endpoint (no auth, signature verified)

Authentication and authorization

All routes related to resources must be protected by authorization. The authentication token is transmitted by the front-end in the HTTP header :

```
Authorization: Bearer <token>
```

Before any modification or deletion of a resource, the server must verify that the identifier of the currently authenticated user matches the identifier of the resource owner. If the identifiers do not match, the request must be rejected with a response :

```
{
    "error": {
        "message": "You are not authorized to modify this resource",
        "message": "403"
    }
}
```

HTTP Code : 403 Forbidden

This verification ensures that only the legitimate owner of a resource is authorized to modify or delete it.

Public Routes

- GET /health
- POST /auth/login
- POST /auth/register
- POST /auth/refresh-token
- POST /auth/users/requester/password
- POST /auth/users/resetter/password

- POST /stripe/webhook (no auth, signature verified)

Protected Routes

All other routes require a valid JWT token.

Routes with ownership verification

The following routes require that the user be the creator of the resource in question or to be an admin :

- All /sessions routes (except admin can access any)
- All /positions routes (except admin can access any)
- All /transactions routes (except admin can access any)
- GET /subscriptions (users see own, admins see all)
- GET /subscriptions/:id (ownership or admin)
- POST/PATCH/DELETE /subscriptions (Admin only)
- GET/PATCH/DELETE /users/:id (ownership or admin)
- PATCH /users/:id/password (ownership or admin)

Session Limits

Session creation is subject to active session limits based on the user's subscription plan:

- **Free users (no subscription)**: 1 active session
- **TRADER plan**: 10 active sessions
- **EXPERT plan**: 30 active sessions
- **Admin users**: Unlimited (bypass all limits)

An active session is defined as a session with `session_status` not equal to `ARCHIVED` (i.e., `RUNNING` or `PAUSED`).

When attempting to create a session that would exceed the limit, the API returns a `400 Bad Request` error with a detailed message:

```
{
    "error": {
        "message": "You have reached your maximum active sessions limit (X sessions for [PLAN_NAME] plan)"
    }
}
```

Note: Archiving a session (setting `session_status` to `ARCHIVED`) frees up a slot for creating new sessions.

Version : 1.3.0
Date : 15 January 2026