

# Mini Prediction Exchange

## User Manual (Trader View)

## 1 Introduction

This document explains how to use the *Mini Prediction Exchange* as a trader. You will learn how to:

- Access the web API interface.
- Create a user account (trader ID).
- Fund your account.
- Create and view markets.
- Place buy orders on open markets.
- Check your balances and open orders.

The backend is running on a FastAPI server with a PostgreSQL database. You will interact with it through an auto-generated web UI.

## 2 Pre-Requisites

### For the Admin (one-time setup)

- The backend server must be running:

```
cd /home/anushka/Documents/Quant/prediction-exchange
source .venv/bin/activate
uvicorn app.main:app --reload
```

- PostgreSQL and Redis are already configured and running.

### For Traders

- You need network access to the machine running the server.
- You need a web browser (Chrome, Firefox, etc.).

## 3 Accessing the API UI

**Step 1:** Open your browser and navigate to: <http://127.0.0.1:8000/docs#/>

If you are on a different machine on the same network, replace 127.0.0.1 with the server's IP address, e.g.:

<http://192.168.1.10:8000/docs>

**Step 2:** You should see the FastAPI *Swagger UI* with a list of endpoints grouped into sections: `users`, `accounts`, `markets` and `orders`.

## 4 Creating a User (Trader ID)

Each trader needs a user account in the system.

**Step 1:** In the `users` section, click on `POST /users`.

**Step 2:** Click on “Try it out”.

**Step 3:** In the *Request body* box, enter your email in JSON format, e.g.:

```
{  
  "email": "alice@example.com"  
}
```

**Step 4:** Click “Execute”.

**Step 5:** Under *Responses*, you should see a 201 status code and a JSON response similar to:

```
{  
  "id": 1,  
  "email": "alice@example.com"  
}
```

The field “`id`” is your **user ID**. Note this number; you will use it in all later steps.

## 5 Funding Your Account

Before placing orders, you must add funds to your account balance. In this prototype we simulate deposits via an API call (dev-only funding).

**Step 1:** In the accounts section, click POST `/accounts/{user_id}/fund`.

**Step 2:** Click “Try it out”.

**Step 3:** In the `user_id` path parameter, enter your user ID (e.g. 1).

**Step 4:** In the *Request body*, specify currency and amount, for example:

```
{  
  "currency": "INR",  
  "amount": 1000  
}
```

**Step 5:** Click “Execute”.

**Step 6:** You should see a response such as:

```
{  
  "user_id": 1,  
  "currency": "INR",  
  "available": 1000.0,  
  "locked": 0.0  
}
```

This means you now have 1000.0 INR of *available* balance.

## Checking Balances

**Step 1:** In accounts, click GET `/accounts/{user_id}/balances`.

**Step 2:** Click “Try it out”.

**Step 3:** Enter your `user_id` (e.g. 1) and click “Execute”.

**Step 4:** You should see a list of balances, for example:

```
[  
  {  
    "id": 1,  
    "currency": "INR",  
    "available": 1000.0,  
    "locked": 0.0  
  }  
]
```

## 6 Creating a Market (Admin / Power User)

Usually only an admin or a trusted user should create markets. A market is a question like: “Will NIFTY close above 25,000 on 31 Dec?” with outcomes like YES and NO.

**Step 1:** In the markets section, click POST /markets.

**Step 2:** Click “Try it out”.

**Step 3:** Fill the *Request body* with a slug (identifier), title, optional description, and outcomes. For example:

```
{  
  "slug": "nifty-above-25000-2025-12-31",  
  "title": "Will NIFTY close above 25,000 on 31 Dec 2025?",  
  "description": "Binary market on NIFTY closing value.",  
  "trading_close_at": null,  
  "settle_at": null,  
  "outcomes": [  
    {"name": "YES", "code": "YES"},  
    {"name": "NO", "code": "NO"}  
  ]  
}
```

**Step 4:** Click “Execute”.

**Step 5:** The response will contain a market id and the created outcomes, e.g.:

```
{  
  "id": 1,  
  "slug": "nifty-above-25000-2025-12-31",  
  "title": "Will NIFTY close above 25,000 on 31 Dec 2025?",  
  "description": "Binary market on NIFTY closing value.",  
  "status": "DRAFT",  
  "trading_close_at": null,  
  "settle_at": null,  
  "outcomes": [  
    {"id": 1, "name": "YES", "code": "YES", "sort_index": 0},  
    {"id": 2, "name": "NO", "code": "NO", "sort_index": 1}  
  ]  
}
```

Note the market id and the outcome IDs. These will be used when placing orders.

## 7 Opening a Market for Trading

A new market starts in DRAFT status. To allow trading, it must be switched to OPEN.

**Step 1:** In markets, click POST /markets/{market\_id}/open.

**Step 2:** Click “Try it out”.

**Step 3:** Enter the market\_id from the previous step (e.g. 1).

**Step 4:** Click “Execute”.

**Step 5:** The response should show "status": "OPEN":

```
{  
  "id": 1,  
  "slug": "nifty-above-25000-2025-12-31",  
  "title": "Will NIFTY close above 25,000 on 31 Dec 2025?",  
  "description": "Binary market on NIFTY closing value.",  
  "status": "OPEN",  
  ...  
}
```

Only OPEN markets can accept new orders.

## 8 Placing an Order

Now that you have:

- a user ID (e.g. `user_id = 1`),
- funds in your account (e.g. 1000 INR),
- an open market (e.g. `market_id = 1`),
- an outcome ID (e.g. `outcome_id = 1` for YES),

you can place a BUY or SELL order.

### Order Inputs

- `user_id`: your user ID.
- `market_id`: the market you are trading.
- `outcome_id`: which outcome (e.g. YES or NO).
- `side`: "BUY" or "SELL".
- `price`: a number between 0 and 1 (e.g. 0.35).
- `quantity`: how many contracts you want.
- `currency`: e.g. "INR".

#### Placing a BUY Order

**Step 1:** In the `orders` section, click POST `/orders`.

**Step 2:** Click "Try it out".

**Step 3:** Fill the body with something like:

```
{  
  "user_id": 1,  
  "market_id": 1,  
  "outcome_id": 1,  
  "side": "BUY",  
  "price": 0.35,  
  "quantity": 10,  
  "currency": "INR"  
}
```

**Step 4:** Click "Execute".

**Step 5:** If you have enough funds, you should see an OPEN order:

```
{  
  "id": 1,  
  "user_id": 1,  
  "market_id": 1,  
  "outcome_id": 1,  
  "side": "BUY",  
  "price": 0.35,  
  "quantity": 10,  
  "quantity_filled": 0,  
  "status": "OPEN",  
  "is_active": true  
}
```

Behind the scenes, the system calculates the required margin (for BUY: `price * quantity`) and moves that amount from available to locked balance.

#### Checking Balances After an Order

**Step 1:** Use GET `/accounts/{user_id}/balances` again.

**Step 2:** Example before trade:

```
{  
  "available": 1000.0,  
  "locked": 0.0  
}
```

**Step 3:** Example after placing the order above ( $0.35 \times 10 = 3.5$ ):

```
{  
  "available": 996.5,  
  "locked": 3.5  
}
```

## Viewing Orders

**Step 1:** In `orders`, click GET `/orders`.

**Step 2:** Optionally fill the `user_id` query parameter (e.g. 1).

**Step 3:** Click “Execute”.

**Step 4:** You will see a list of your orders, with their status and details.

## 9 Using Real-World Data (Conceptual)

Currently, markets are created manually by the admin via POST `/markets`. To make these markets track real events (e.g. stock indices, sports scores):

- The admin chooses real-world events (e.g. NIFTY close, match winner) and creates markets whose `slug`, `title` and `settle_at` correspond to those events.
- A separate background process or script (to be implemented) can connect to external data sources (stock exchanges, sports APIs, news feeds) to:
  - Update market status (e.g. automatically close trading at a deadline).
  - Resolve outcomes once the real event is known (e.g. YES wins).
- In this initial version, resolution and status changes can be done manually by the admin; the order placement and balance logic already work as they would in a live system.

## Summary

For traders, the basic workflow is:

1. Open `http://127.0.0.1:8000/docs#/docs` in your browser.
2. Create a user via POST `/users`.
3. Fund your account via POST `/accounts/{user_id}/fund`.
4. Ask the admin for available market IDs and outcome IDs, or view them via GET `/markets`.
5. Place orders via POST `/orders`.
6. Track your balances and orders via the `accounts` and `orders` endpoints.

As more features are added (matching engine, order cancellation, automatic market resolution, and external data feeds), this interface can continue to serve as a convenient way to interact with the exchange in real time.