Blofin API

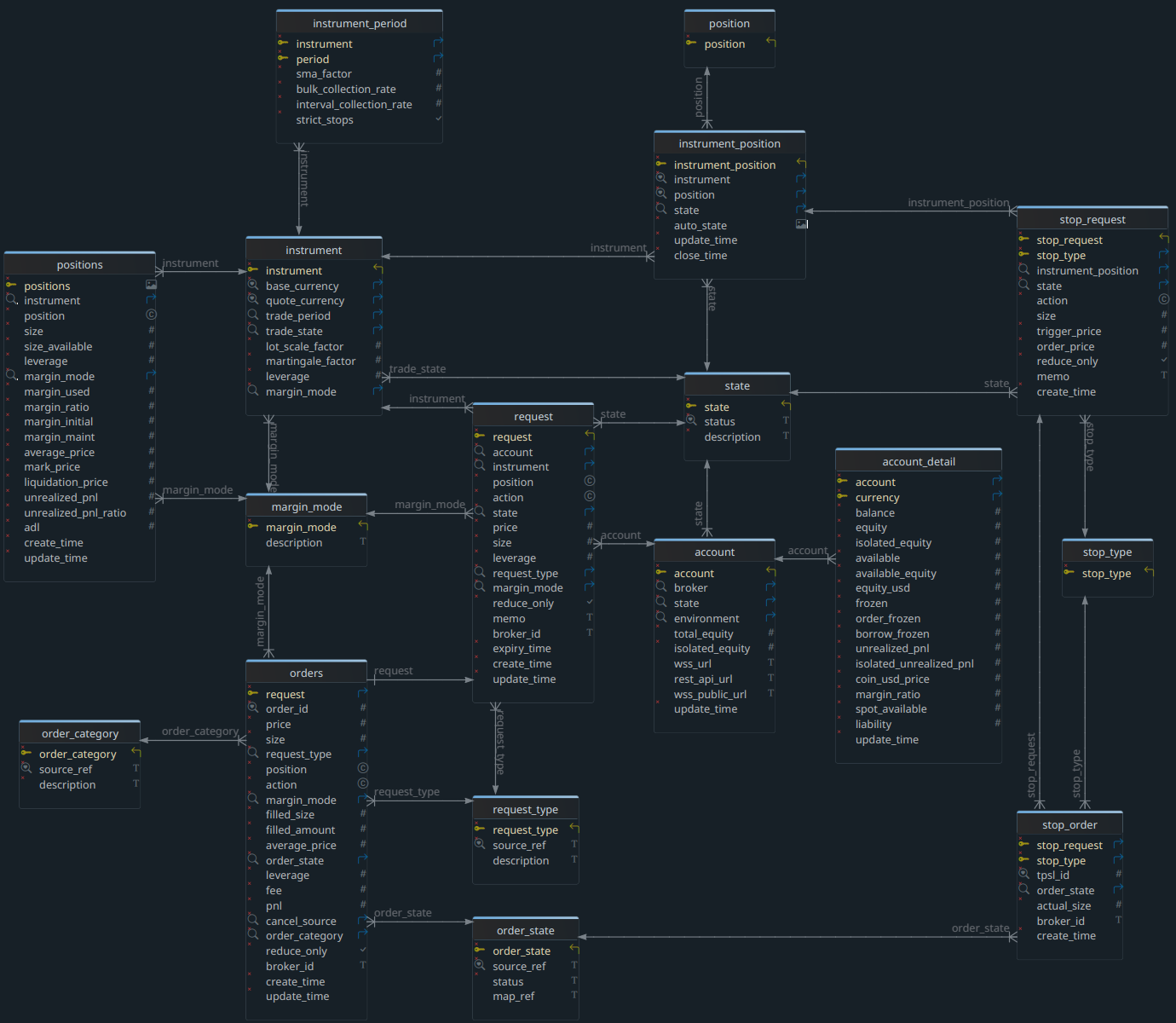
Full-Regression Test Protocol

*August 10, 2025*

# Test Protocol Setup & Configuration

## Database Preparation

The case studies examined in this API test protocol implement the database entities displayed in figure 1. The instructions outlined in this section establish a data baseline to be used to determine the success and/or failure of each case study executed.

figure 1.

### Application Operation Overview

This section outlines basic procedures for operating the Blofin API application. This procedure shall remain in effect until the development of a graphical user interface (gui) is completed. This procedure is executed from a command line interface (cli) typically, but not limited to, the terminal included with VS Code. For this example, the procedures shall be executed with linux Konsole.

#### Environment Setup & Configuration

Open a terminal window. If/when prompted for the ssh password, enter the value provided to you by the system administrator. Given a correctly configured test environment, the node.js version shall be displayed in the console as shown in figure 2. Confirm that this node.js version is compatible for use in the test environment before proceeding.

Next, navigate to the application root folder with the following command:

cd Projects/blofin-data

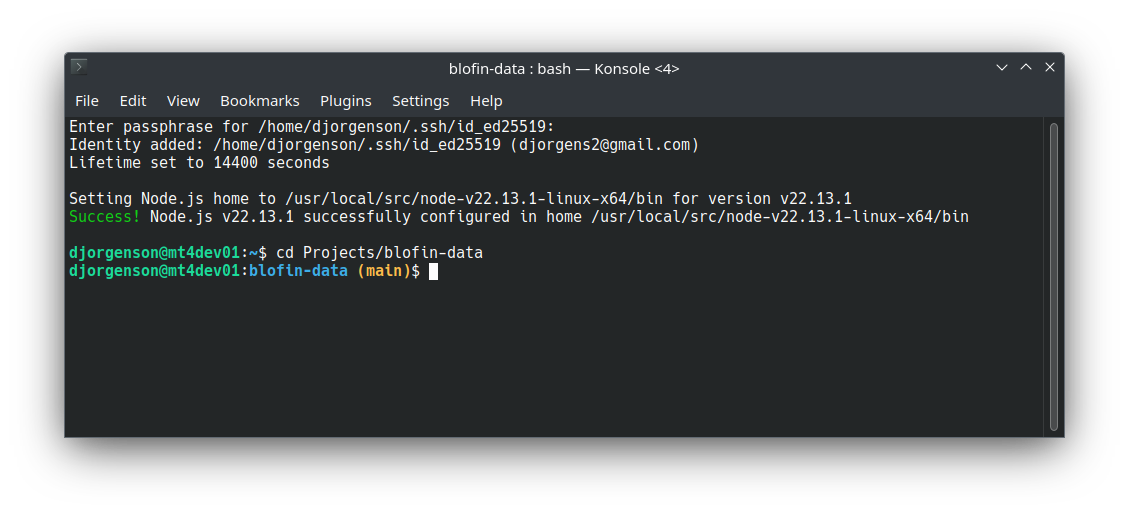


figure 2.

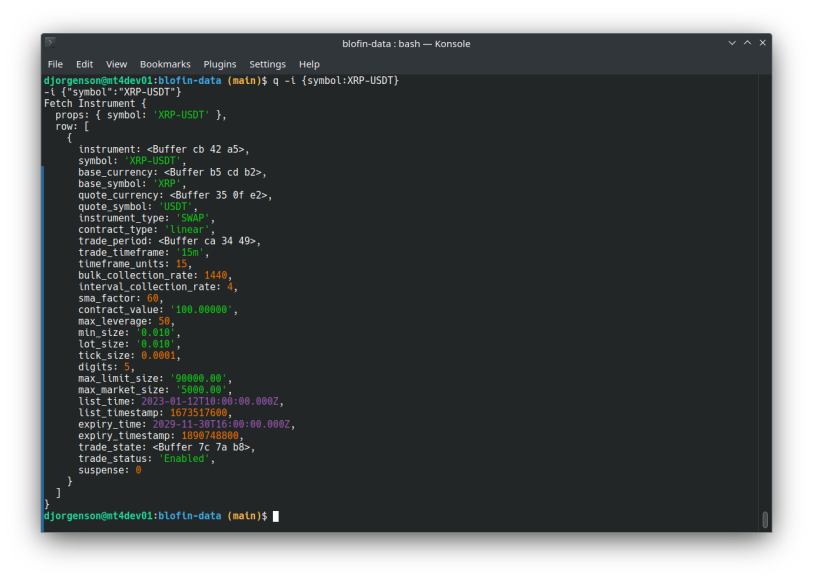
#### Select Test Symbols (instruments)

For the purposes of this full-regression test protocol, a minimum of two symbols (called ‘instruments’ in Blofin parlance) are required to identify behavioral issues by trade state; i.e., symbols that are ‘Enabled’ for trading, and symbols that are in ‘Disabled’, ‘Halted’, or ‘Suspended’ trade states operate on code specific to the state of the symbol.

From the console using the q.sh query tool, verify the trade state of the symbols to be tested in this protocol by executing the following commands or a variation thereof as follows:

q -i {symbol:XRP-USDT}

q -i {trade\_status:Enabled}



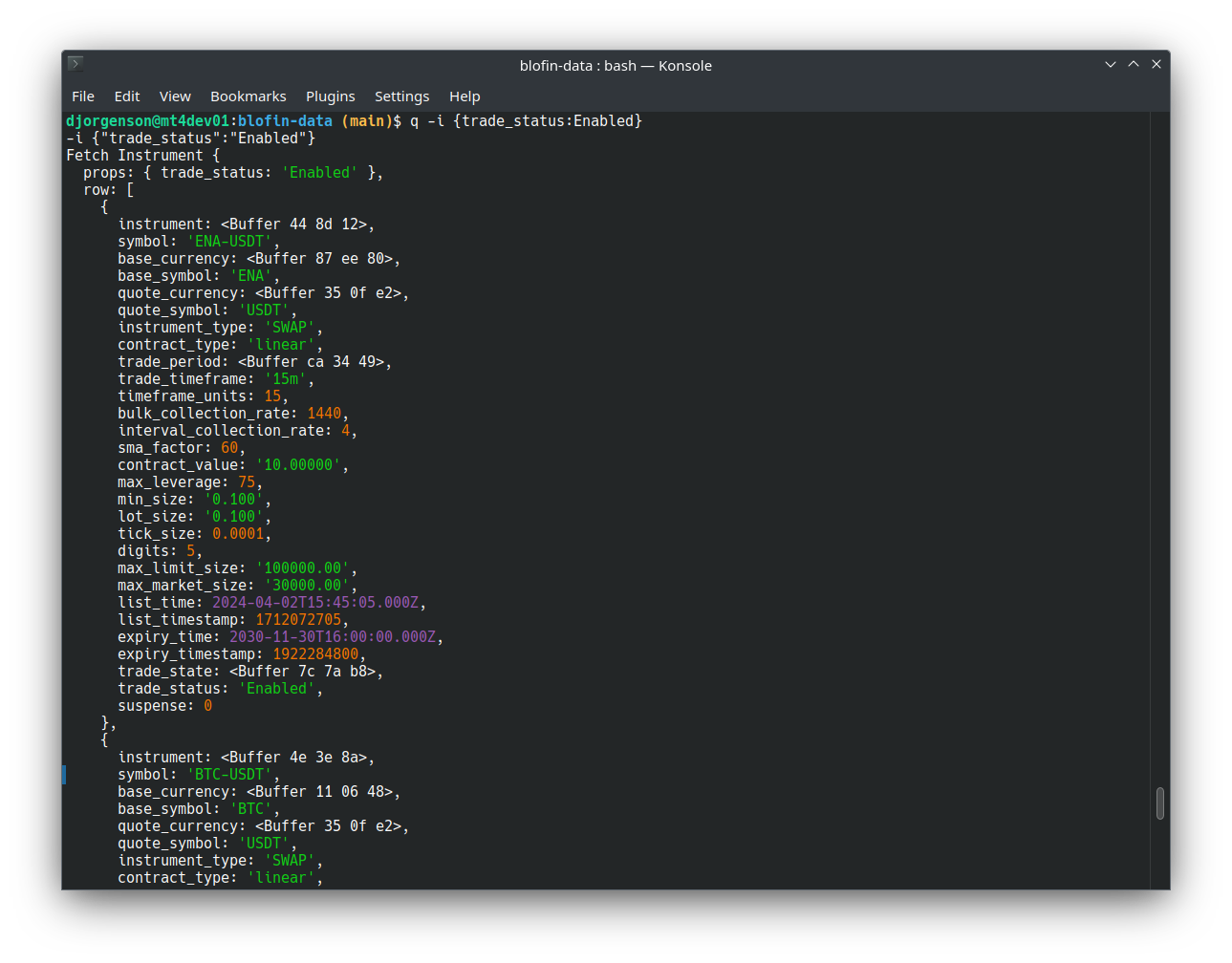


figure 3.

As seen in the figure 3, q.sh tool query results can be filtered by any named field[[1]](#footnote-0).

#### Application Startup

Using the following command, start the application and log sysout to a logfile. This command launches the application and returns the process id (PID) of the execution running in the background as seen in figure 4.

npx tsx app > app.log &

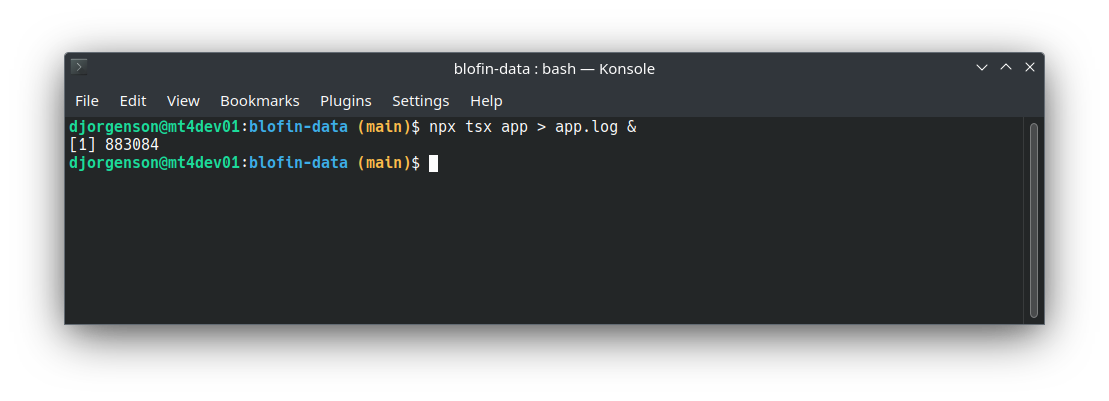


figure 4.

#### Application Monitoring

Application operations are written to the logfile stated when the application was started; each operation, success, and failure is continuously streamed to the logfile. To monitor the process, open the logfile with a text editor or echo the content to the console, whichever means works best. In the example below, the process continuously displays to the console the latest instructions performed as they are executed. To set up continuous monitoring, use the following command:

tail -f app.log

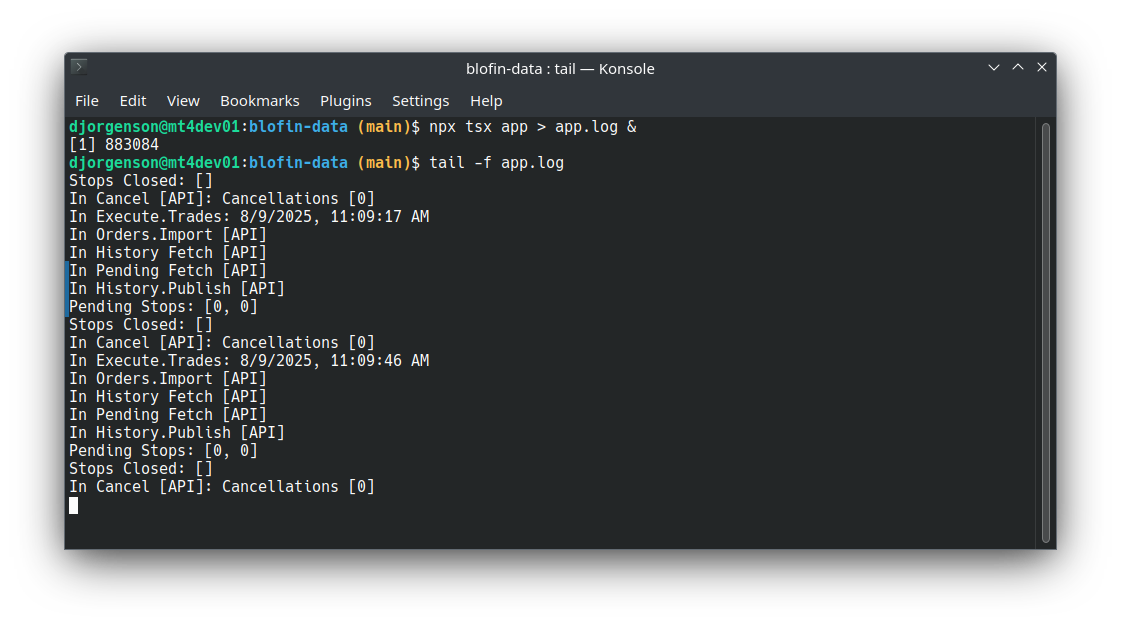


figure 5.

Once monitoring is started, the latest instructions executed are appended to the log and displayed on the console. To end monitoring, press <ctrl+c>. While testing, all logfiles shall be retained and reviewed to ascertain the success or failure of each test executed within this protocol.

#### Application Shutdown

To stop the application, first identify the PID. If not recorded on startup, execute the following command:

ps -ef | grep -v grep | grep "tsx"\ "app"

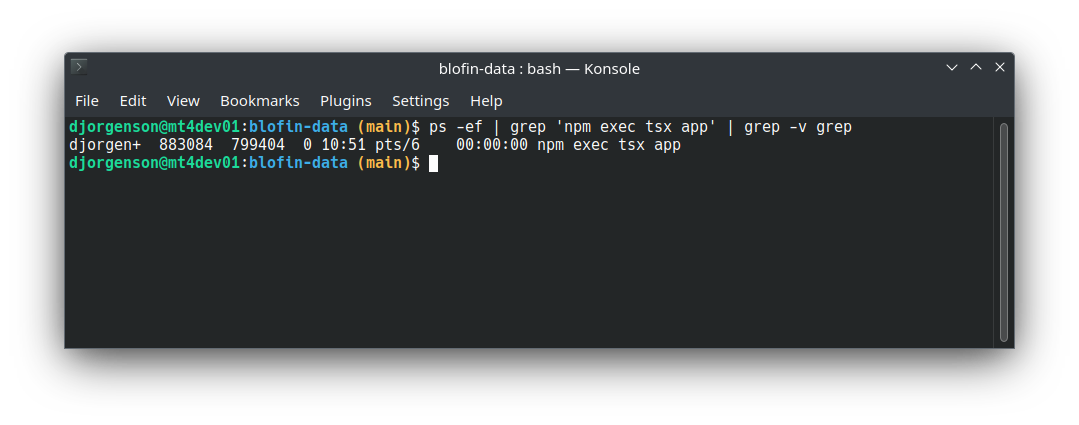


figure 6.

Next, terminate the application with the following command using the PID of the currently running process as follows:

sudo kill -9 883084

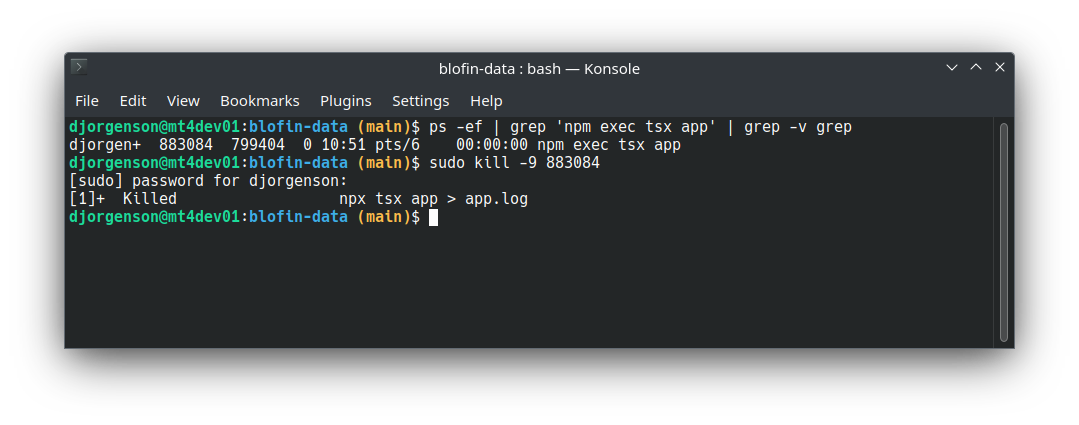


figure 7.

### Data Baseline Setup

Prior to executing this test protocol, a baseline dataset is essential to determine the outcomes of test results. This set will vary based on previous historical data retrieved from Blofin. The steps in this section shall be executed prior to any/all tests conducted within this protocol.

#### Stop Application Process

To prevent database updates while establish the data baseline, the application system shall be quiesced. To stop the application process, refer to Section 1.1.1.5 Application Shutdown. Once the system is successfully quiesced, proceed to the next section. To confirm the state of the application, executing the following command shall report no running processes.

ps -ef|grep -vE "firefox|browse|brave|root|NX|grep"|grep -E "node|npm|tsx|mjs|ipc"

#### Remove Prior Data

From a SQL client, connect to the MySql blofin database and execute the SQL statements from table 1.

|  |
| --- |
| **Data Baseline** |
| delete from blofin.orders; |
| delete from blofin.request; |
| delete from blofin.stop\_order; |
| delete from blofin.stop\_request; |
| delete from blofin.positions; |

Table 1

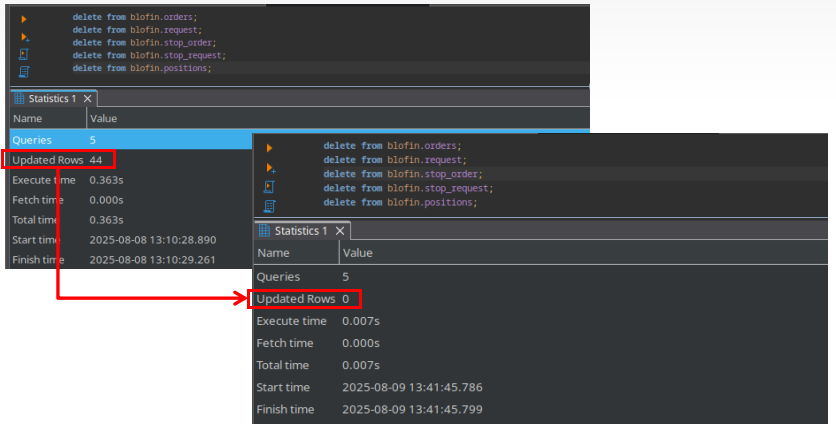
The number of rows removed may vary after each execution; the purpose of this is step is to remove all rows from prior tests and achieve a zero rows of prior operational data.

figure 8.

#### Verification

Upon successful completion of the previous step, from a SQL client, connect to the MySql blofin database and execute the SQL statements from table 2.

|  |
| --- |
| **Data Baseline** |
| select \* from blofin.orders; |
| select \* from blofin.request; |
| select \* from blofin.stop\_order; |
| select \* from blofin.stop\_request; |
| select \* from blofin.positions; |
| select \* from blofin.vw\_orders; |
| select \* from blofin.vw\_requests; |
| select \* from blofin.vw\_stop\_orders; |
| select \* from blofin.vw\_stop\_requests; |
| select \* from blofin.vw\_positions; |
| select \* from blofin.vw\_api\_requests; |
| select \* from blofin.vw\_api\_stop\_requests; |

Table 2

These queries cited in Table 2 include both tables inserted to and modified by the application data interchange with Blofin API and all of the views governing the access to this data. A successful result is achieved when all of the objects queried from table 2 contain zero rows. Repeat this test over five or more minutes to ensure that a full processing cycle ( 30s ) has completed as shown in figure 9.

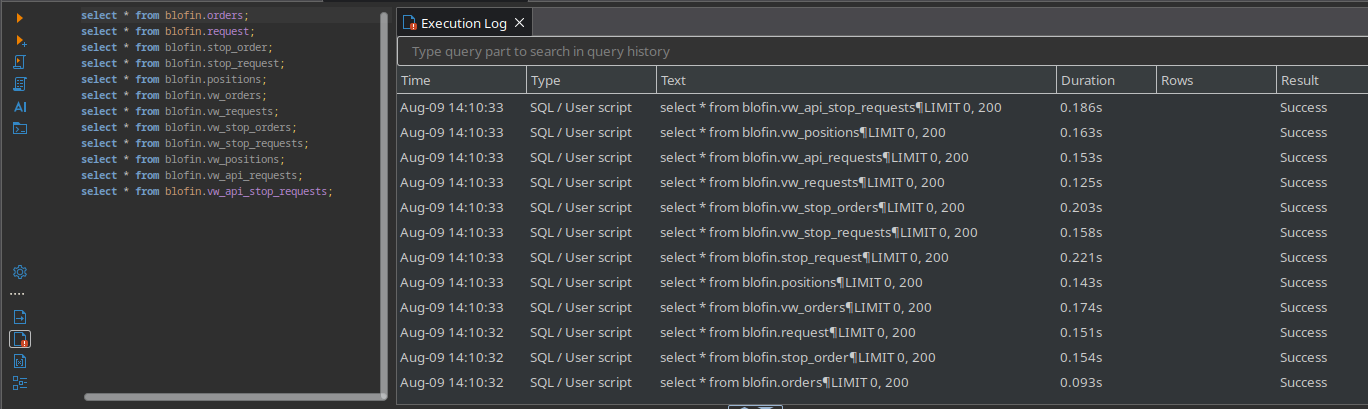


figure 9.

#### Application Start

Next, start the application and allow for at least one processing cycle ( 30s ) to complete. To start the application, execute the instructions detailed in Section 1.1.1.3 Application Startup. Upon successful startup, the application synchronizes the local database with historical data retrieved from Blofin (shown in the logfile below). Having reached this point, the data captured represents the baseline; all data changes shall be reviewed as part of this effort.

Subscriptions: { channel: 'account' }

Subscriptions: { channel: 'positions' }

Subscriptions: { channel: 'orders' }

In WSS.Publish [API]

Instruments Suspended: 0 []

Instruments Updated: 0 []

In Execute.Trades: 8/9/2025, 4:24:46 PM

In Orders.Import [API]

In History Fetch [API]

In Pending Fetch [API]

In History.Publish [API]

Orders Imported: 20 [

{

request: <Buffer 00 f7 2e 43 67 2f>,

order\_id: 1000109643347,

instrument: <Buffer cb 42 a5>,

price: 3.5,

size: 100,

request\_type: <Buffer 6e b6 c5>,

action: 'sell',

position: 'short',

margin\_mode: 'cross',

filled\_size: 0,

filled\_amount: undefined,

average\_price: 0,

order\_state: <Buffer 7d 5c 89>,

leverage: 10,

fee: 0,

pnl: 0,

order\_category: <Buffer 85 e8 4f>,

cancel\_source: <Buffer 4e 8b 73>,

reduce\_only: false,

broker\_id: undefined,

create\_time: 1754609037377,

update\_time: 1754609182240

},

...

#### Capture Baseline Dataset

Upon completion of the first processing cycle from the prior step the local database is now synchronized with Blofin historical data. This step captures the data in an offline file which, for purposes of this exercise, shall be a Microsoft compatible spreadsheet called WPS; an option that best serves linux development environments. The data extraction tool used in this exercise is called DBeaver which also integrates seamlessly with linux. To begin the capture, open a new worksheet and a DBeaver session connected to the the blofin schema.

Table 3 below lists the table (physical) and view (logical) targets to be extracted. Each query shall be executed, exported, then imported into our baseline dataset. Begin with the first query, execute all steps through the import, then execute the next query; repeat the process query by query until finished.

|  |
| --- |
| **Data Baseline** |
| select \* from blofin.request order by request; |
| select \* from blofin.orders order by request; |
| select \* from blofin.stop\_order order by stop\_request, stop\_type; |
| select \* from blofin.stop\_request order by stop\_request, stop\_type; |
| select \* from blofin.positions order by positions; |
| select \* from blofin.vw\_requests order by request; |
| select \* from blofin.vw\_orders order by request; |
| select \* from blofin.vw\_stop\_orders order by stop\_request, stop\_type; |
| select \* from blofin.vw\_stop\_requests order by stop\_request, stop\_type; |
| select \* from blofin.vw\_positions order by positions; |
| select \* from blofin.vw\_api\_requests order by clientOrderId; |
| select \* from blofin.vw\_api\_stop\_requests order by clientOrderId; |

Table 3

##### Data Selection

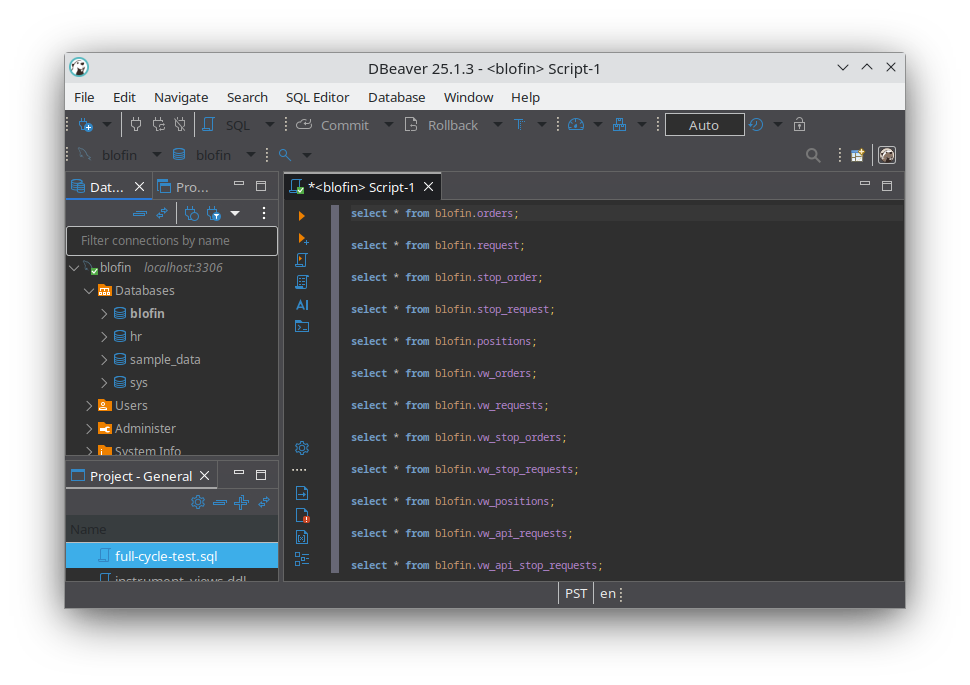
Using DBeaver, create a new SQL script (ctrl+] or from the SQL Editor menu), select the queries from Table 3 and copy (ctrl+c) and paste (ctrl+v) the queries into the new script.

figure 10

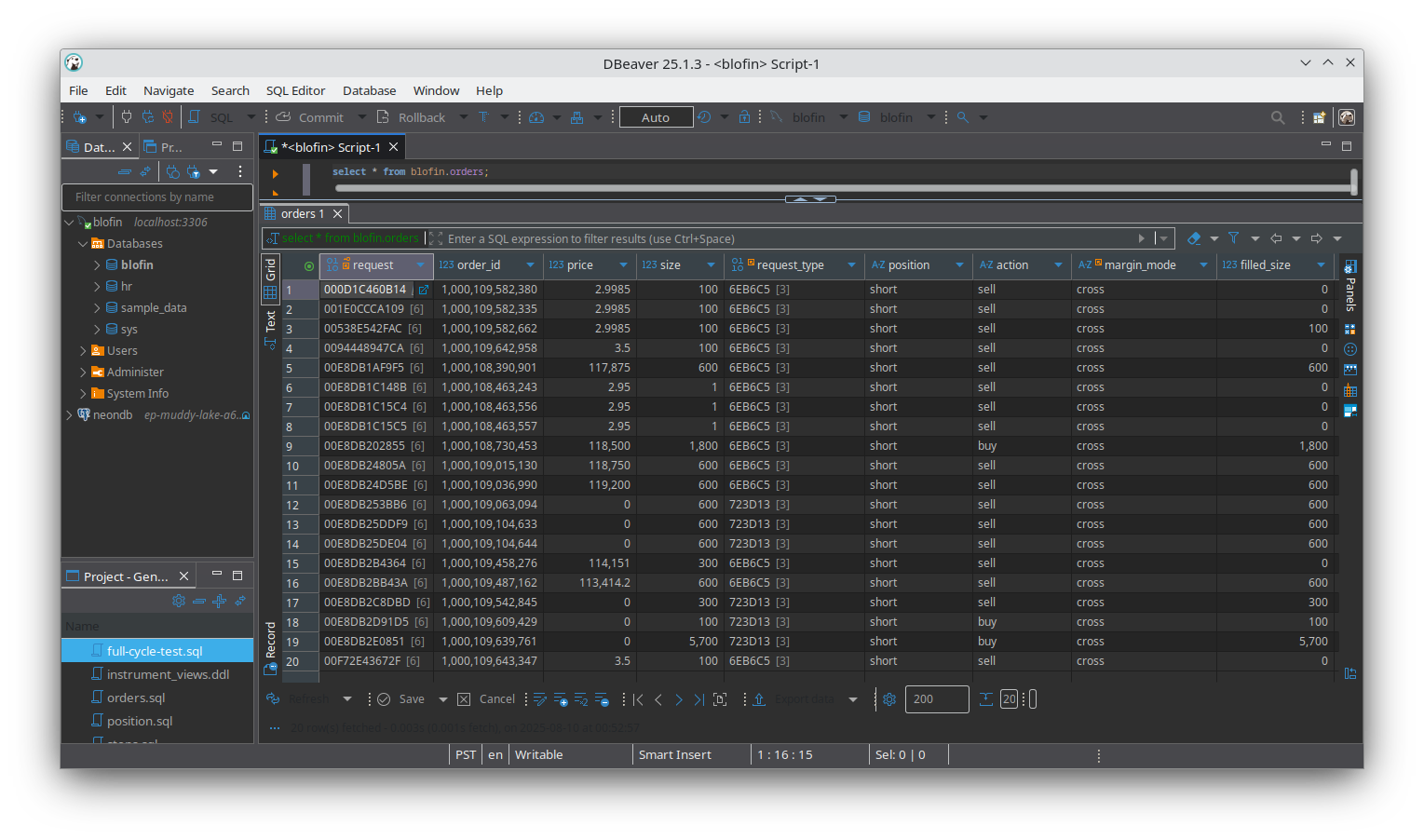
With the mouse, left click on the first query then press <ctrl+enter> on the keyboard. Done correctly and given the amount of recent historical data retrieved from Blofin, results similar to those shown in figure 11 may appear.

figure 11

##### Data Export

In DBeaver, click on any cell in the results pane, press <ctrl+a> to select all the data, then press <ctrl+c> to copy the data. In the unlikely event that there are more than 200 rows of data (the default fetch size on result sets), press <shift+ctrl+=> to fetch the remaining rows before selecting and copying the data.

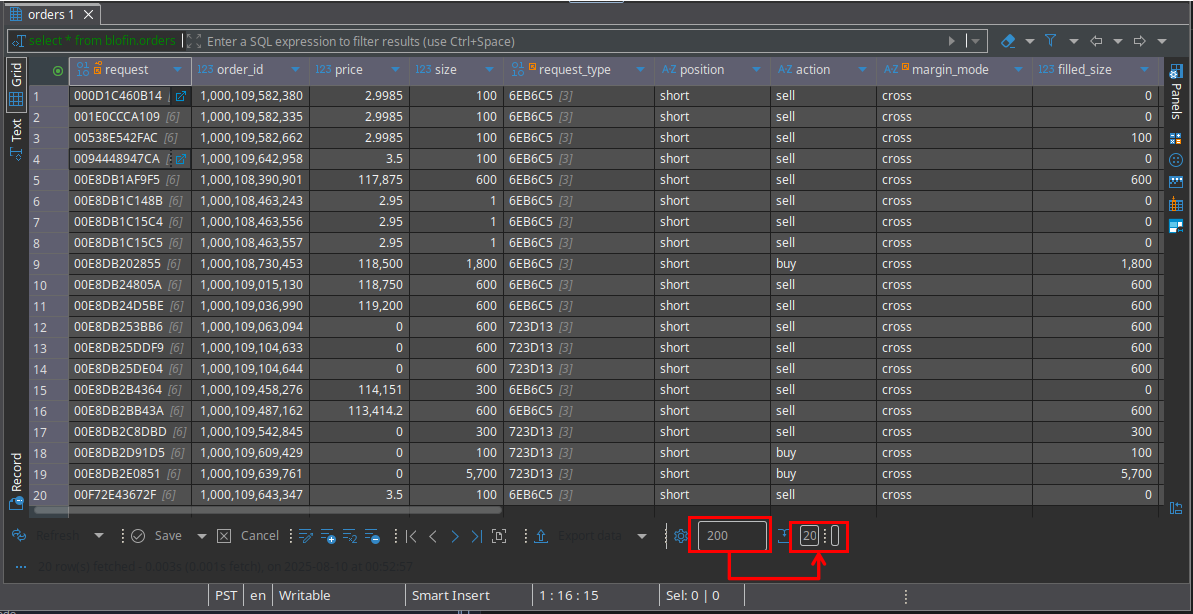


figure 12

##### Data Import

In the spreadsheet, left-click on cell [A2] and press (ctrl+v) to paste the resultset. As seen in figure 13, the sheet tab was renamed to the name of the query source (‘subject area’) and column headers were added. In this example, the completed data capture for the blofin.orders table is presented.

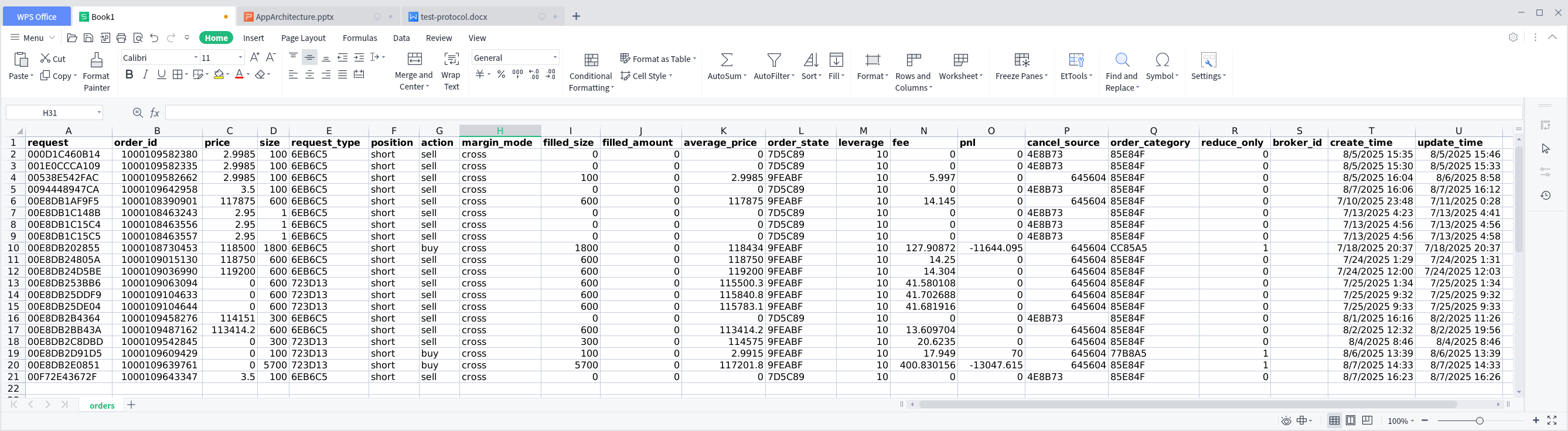


figure 13

##### Completing the Data Baseline

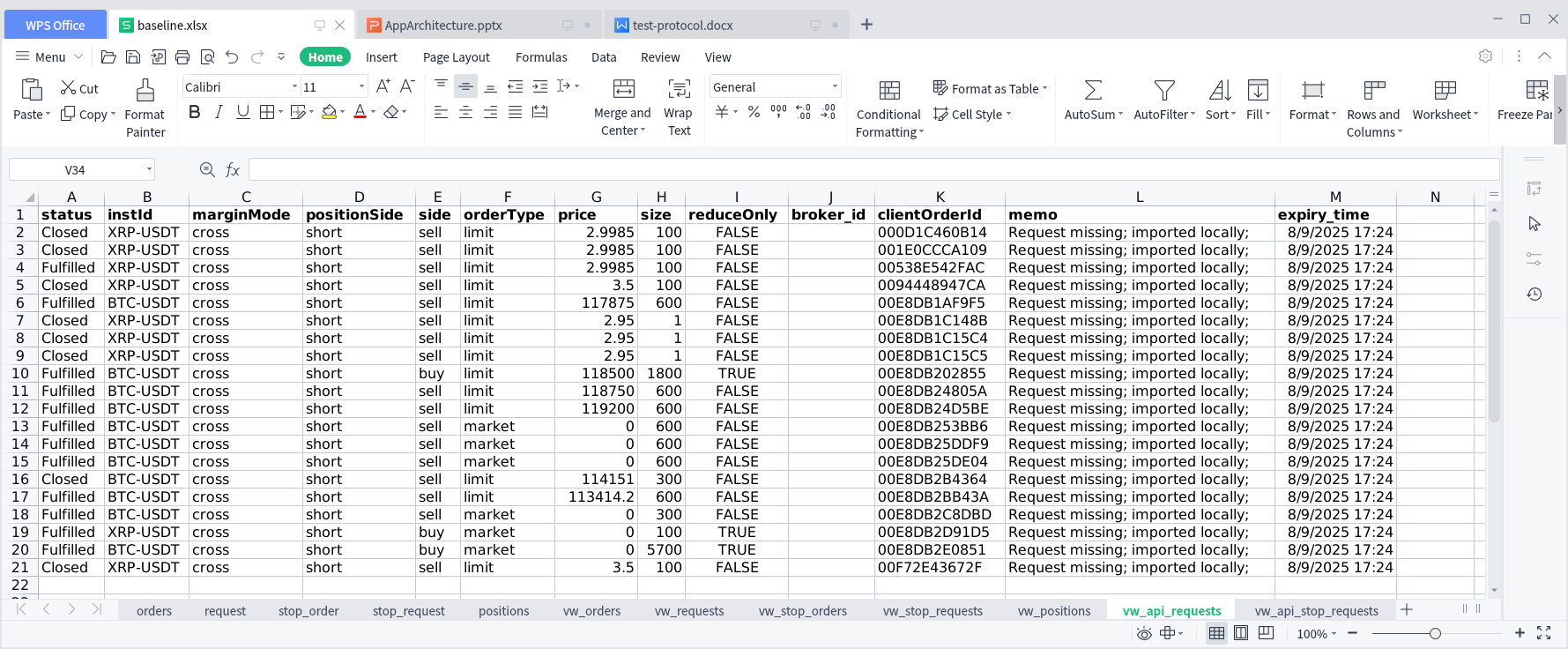
Next, repeat this process for each query to complete the data baseline. A sample data baseline spreadsheet is shown in figure 14. Prior to the execution of any of any of the test protocols documented in this section, the Tester shall create a copy of the baseline.xlsx spreadsheet file using a naming convention specific to the test executed.

figure 14

### Test Execution General Principles

This Test Protocol is organized into case studies representing specific application functional subject areas and may consist of one or more independent tests. Each study, and any of its sub-studies, consists of a defined, detailed set of instructions that, when executed as documented, produce material, results, and analysis to be used to determine the success or failure of the test. This section outlines the general approaches, tools and methods used within each test scenario.

#### Environment Preparation

If not already present, create a folder in the Project/blofin-data/fcrt folder along with any sub-folders named after the section headings defined in this document. The example shown in figure 15 depicts the folder structure to be implemented for the first study of the Section 2.1 Command Line Interface (CLI) Order Entry.

figure 15

##### Documents

The documents folder contains the test protocol procedure document and verifcation worksheet checked out from the version control repository, the data baseline worksheet produced in Section 1.1.2 Data Baseline Setup and a copy of the data baseline worksheet to be used to compare with the baseline. Any documents remaining from a prior test protocol execution shall be removed before executing a revised or otherwise new test protocol.

##### Logs

The logs folder contains the application logs for the existing test protocol execution. This folder shall be emptied prior to the execution of a revised or otherwise new test protocol.

Log files are created from the application, from the test protocol module, and possibly from the console if an event occurs either internally that logs a message to the console or in the event of a catastrophic failure wherein the application ends abnormally.

##### Test

The test folder contains the typescript application components to be executed during the test protocol. This folder may include one or more files including application executables, typedefs, or special application components that shall be checked out from the version control repository.

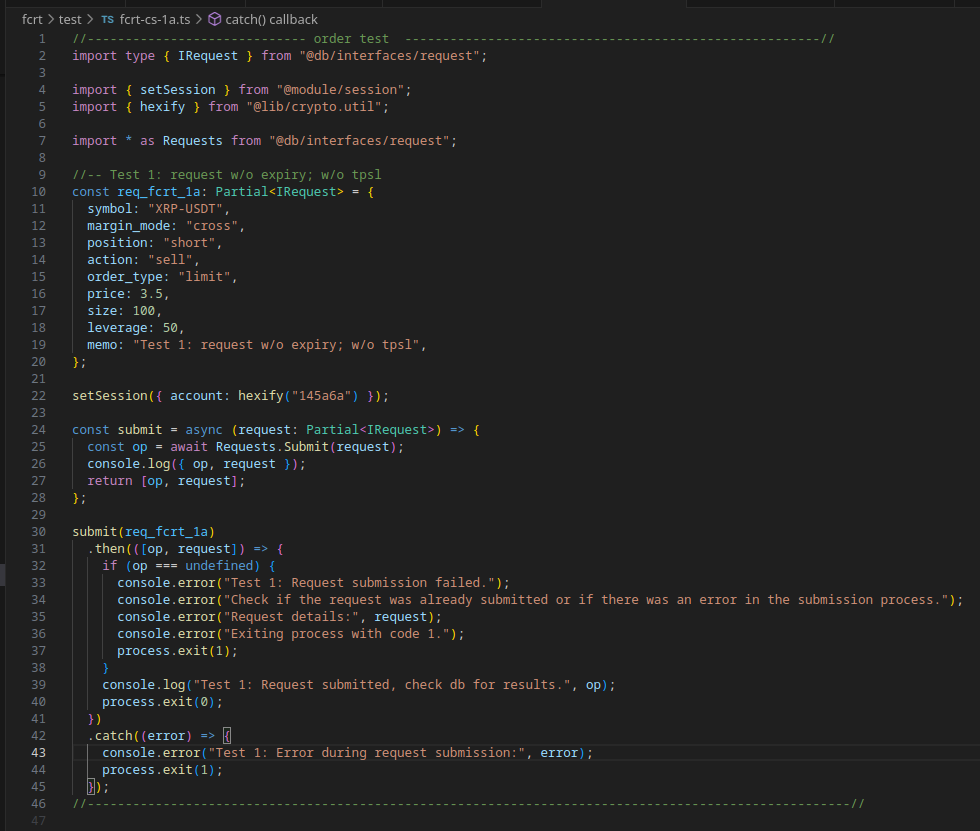
# Case Study: Order Management

This section comprises the studies directly concerning the application’s order management components and process integration with the Blofin API endpoints. The principle objective of these studies is to ensure the successful operation of the application given properly formatted and valid input. Studies presented later in this document shall address data quality issues; whereas an ‘out-of-box’ study consists of unknown or otherwise randomized inputs without expected, predefined outputs, the studies defined in this section are ‘in-the-box’, i.e., implemented using known inputs to derive specific, expected outputs.

## Command Line Interface (CLI) Order Entry

A CLI Order Entry test emulates the automated processing behavior of the internal application. These tests implement the interfaces the application uses to place, update, and cancel requests ( pre-submission ) and orders ( post-submission ). Test results shall be maintained in the worksheet named in each sub-section.

### Request without expiry



### Request with Expiry

//------------------------ order test ----------------------------//

import { setExpiry } from "@lib/std.util";

import { setSession } from "@module/session";

import { IStopOrder, Submit } from "@db/interfaces/stops";

import { hexify } from "@lib/crypto.util";

import \* as Requests from "@db/interfaces/request";

import \* as Instruments from "@db/interfaces/instrument";

//-- test 2; request w/ expiry & no tpsl;

const request: Partial<Requests.IRequest> = {

symbol: "XRP-USDT",

margin\_mode: "cross",

position: "short",

action: "sell",

order\_type: 'limit',

price: 3.5,

size: 100,

leverage: 50,

expiry\_time: setExpiry("30m"),

};

setSession({ account: hexify("145a6a")})

const submit = Requests.Submit(request);

console.log({submit, request});

1. See Appendix A for a more complete description on the usage of the q tool. [↑](#footnote-ref-0)