1. User Story

1.1. As a LiteFinance user, I want to be able to place a market order of type "sell" through the web application so that I can sell my asset at the current market price.

1.2. As a LiteFinance user, I want to be able to place a market order of type "sell" through the mobile application so that I can sell my asset at the current market price.

1.3. As a LiteFinance user, I want to be able to place a market order of type "buy" through the web application so that I can buy an asset at the current market price.

1.4. As a LiteFinance user, I want to be able to place a market order of type "buy" through the mobile application so that I can buy an asset at the current market price

1.5. As a LiteFinance user, I want to receive the execution status of my market order of any type via push notification on my phone so that I can be sure the trade went through without errors.

** separated 2 types of market orders (buy/sell), since the request may have different attributes and checks.*

2. Functional Requirements (I will describe just scenario with market order type = buy)

Use Case1

Description (purpose):

The actor (user) can place a market order of type "buy" through both the web and mobile applications.

Preconditions:

1. Trading day is open in the system.
2. The user is verified in the system
3. The user has enough money on his balance to buy.

Postconditions:

1. The user has purchased the asset at the current financial value.

Main Flow:

1. The user navigates to the LiteFinance login page.
2. The system displays login and password fields.
3. The user completes verification by entering login and password.
4. The user selects which asset he wants to buy and specifies that it will be a market order.
5. The user specifies the quantity of the asset he intends to buy.
6. The user specifies the trade direction = "buy" and click "OK".
7. The system created a new order in status =new
8. The system processes the request and matches it with the best available prices (Order_status = in progress).
9. The system checks that the user has enough money on his balance

10.The system executes the purchase (Order_status = Processed).
11. The user sees the new trade in his portfolio, but at the actual execution price

Alternative Flows:
2a. The user has not previously registered in the personal account:
1. The system prompts the user to register.
2. If the user refuses, the scenario returns him to step 2 of the Main Flow.
3. If the user agrees, the "Register in Personal Account" use case is initiated.

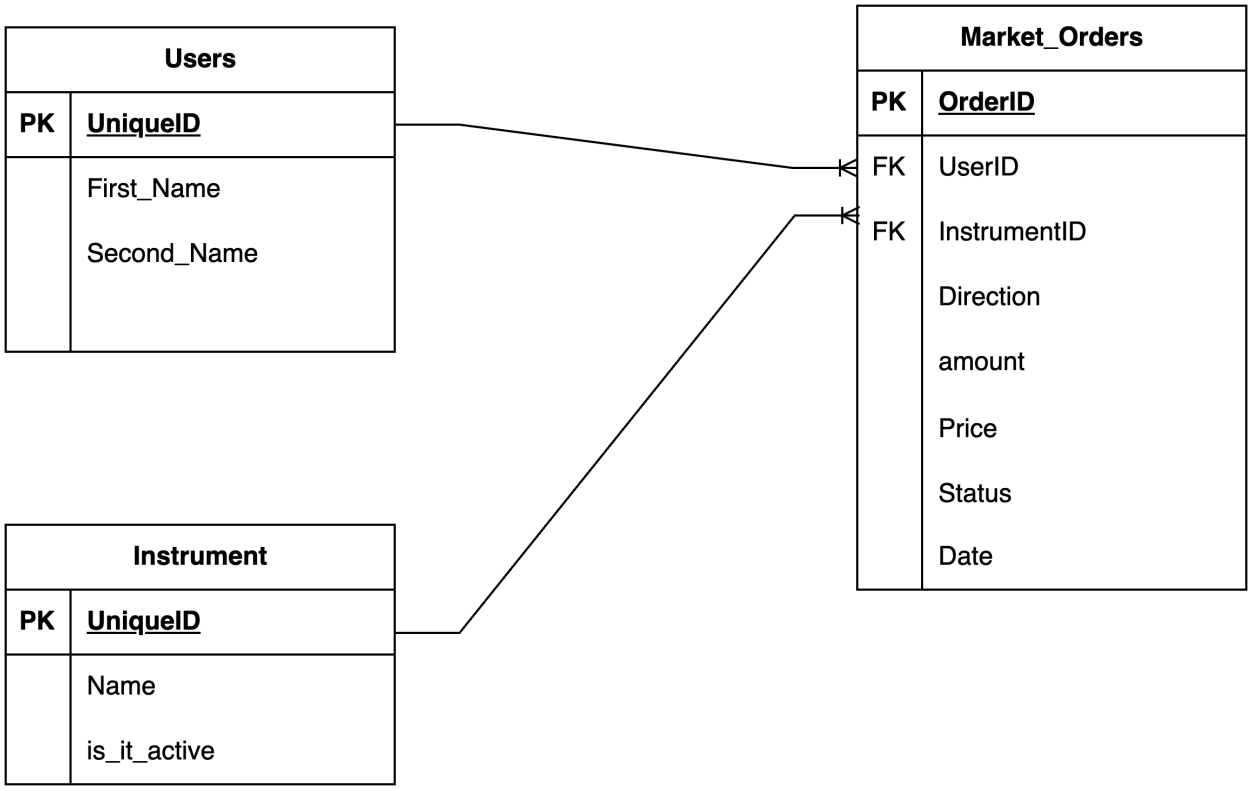
2б The user enters incorrect login details:
1.The system prompts to re-enter the data.
2. If the user refuses, the scenario returns him to step 2 of the Main Flow.

8a. The user does not have enough money to buy:
1. The system cancels the market order (order_status = cancel).
2. The user sees his market order in the status "Cancelled" with the reason "insufficient money on balance".

Exceptions / Interruptions::

1. The system allows the user to return to the previous step at any point before step 8.
2. The system allows the user to terminate the process at any step before step 8.
3. If the user is inactive for 2 minutes before step 8, the system shows the message “Are you still with us?” and cancels the order if inactivity continues for another 2 minutes.

ER-Diagram



3. Non-Functional Requirements

3.1. Performance
The system must be capable of handling peak loads (must be clarified).
Target metric – *n* market orders per second.

3.2. Security
The system must support user authentication by login (=email) and password.

3.3. Scalability
Growth of market orders up to *n* per second (must be clarified).

3.4. Reliability
In case of a system failure, recovery time should not exceed 5 minutes in 90% of cases.

3.5. Compatibility
Mobile app must be compatible with iOS 15+ and Android 10+.
Web app must correctly display all UI elements and perform core functions on desktop operating systems Windows 10+ and macOS 11+, as well as on Android 10+ and iOS 15+ mobile devices, in the latest versions of Chrome, Firefox, and Safari.

3.6 Maintainability
System actions must be logged. Logs must be viewable through a graphical interface.

4. System Diagram (see attached pnd file named "Sequence diagram"

5. Backlog Task
— Title: Create market order in Mobile
— Description:
As a LiteFinance user, I want to be able to place a market order of type "buy" through the mobile application so that I can buy an asset at the current market price.
Functional and Non-functional requirements - see here (link to the description)
API description and Sequence diagram - see here (link to the description)
— Acceptance criteria:
1. The application provides the ability to select a specific financial asset to purchase.
2. The user can specify the desired volume of the asset to buy.
3. The user can specify the order type = "market".
4. The order must be executed immediately at the best available market price.
5. After execution, the user must receive a push notification on his phone that the trade was completed, including the purchase price and volume.

— Priority (e.g. High, Medium): clarify with product owner or stakeholder
— Estimation (in hours or story points): calculate together with team (developers and testers)
— Tags (e.g. mobile, api, backend, trading): mobile, api, backend

6. Test case (POST /orders)
Positive scenario 1
1.Priority: High

2. Name: add a new market order
3.Preconditions:
3.1 User has been authorisation
3.2. Content-type: application/json
4.Test data:
4.1 instrument = currency
4.2 direction = buy
4.3 amount = 100
RequestBody:
{"instrument": "currency",
"direction": "buy",
"amount": 100
}
5. Steps:
5.1 Initiate a service call using the post method with body
Expected result: the request was successfully initiated
5.2 Check status code
Expected result: HTTP Status: 200 OK
5.3 Check response body
Expected result: {"orderId": 123}
Negative scenario 1
1.Priority: High
2. Name: NO access to add a new market order
3.Preconditions:
3.1 User has not been authorisation
3.2. Content-type: application/json
4.Test data:
4.1 instrument = currency
4.2 direction = buy
4.3 amount = 100
RequestBody:
{"instrument": "currency",
"direction": "buy",
"amount": 100
}
5.1 Initiate a service call using the post method with body
Expected result: the request was invalid
5.2 Check status code
Expected result: HTTP Status: 403 access denied
Negative scenario 2:
1.Priority: Medium
2.Name: invalid request
3.Preconditions:
3.1 User has been authorisation
3.2. Content-type: application/json
4.Test data:
4.1 instrument = currency
4.2 direction = buy
4.3 amount = abs
RequestBody:
{"instrument": "currency",
"direction": "buy",

"amount": abs
}
5.1 Initiate a service call using the post method with body
Expected result: the request was invalid
5.2 Check status code
Expected result: HTTP Status: 400 invalid request