



TUNIS BUSINESS SCHOOL

YASSINE BARKIA

DR. MONTASSAR BEN MESSAOUD

WEB SERVICES PROJECT

Digital Finance API: ForexWar_TN

January 23, 2024

Abstract

The ForexWar_TN API project introduces an educational platform for simulated currency trading in Tunisia, addressing the lack of practical exposure to financial markets. Grounded in extensive research, the project employs FastAPI and SQLAlchemy, integrating with the fastFOREX API for real-time currency data. Utilizing Visual Studio Code, OAuth2, and Insomnia for development, security, and testing, the API fosters hands-on learning. Future enhancements include profit calculation, Alembic for database migration, front-end integration, deployment planning, and rigorous testing. This initiative signifies a pivotal step in demystifying financial markets, providing a valuable tool for experiential learning in the Tunisian context.

Keywords— trading, forex, API, fastAPI, oauth2

Declaration of Plagiarism

1. I understand that plagiarism means taking and using the ideas, writings, works, or inventions of another as if they were one's own. Plagiarism includes verbatim copying and the extensive use of another person's ideas without proper acknowledgment, which includes the proper use of quotation marks. Plagiarism covers the use of material found in textual sources and from the Internet.
2. I acknowledge and understand that plagiarism is wrong.
3. I understand that my research must be accurately referenced. I have followed the general rules and conventions concerning referencing, citation, and the use of quotations.
4. This assignment is my own work, or my group's own unique group assignment. I acknowledge that copying someone else's assignment, or part of it, is wrong, and that submitting identical work to others constitutes a form of plagiarism.
5. I have not allowed, nor will I in the future allow, anyone to copy my work with the intention of passing it off as their own work.

Contents

1	Introduction	1
1	Context	1
2	Motivation	1
2	Methodology	3
1	Research	3
2	Benchmarking	5
2.1	MyInvestia:	5
2.2	Investopedia Simulator Game:	6
2.3	MetaTrader4 Demo Account:	6
3	Rules Design	7
3	Implementation	9
1	Architecture	9
1.1	Use Case Diagram	9
1.2	Class Diagram	9
2	RDBMS Choice	9
3	Main Tools & Technologies Used	12
3.1	Visual Studio Code	12
3.2	FastAPI	12
3.3	SQLAlchemy	12
3.4	Pydantic library:	13
4	Initiation	13
5	Security Implementation	13
5.1	Authorization Router	13
5.2	oauth2.py file	14
6	Profile Router	14
7	Trading Router	15
8	Currencies' Router	15
8.1	fastFOREX API integration :	15
8.2	Get_current_exchange_rates:	16
8.3	Get_time_series:	16
9	Testing	16
10	Future Enhancements and Recommendations	17

4 Conclusion and Perspectives	18
A Database	20
B Code Snippets	22

List of Figures

2.1 Types of Market Orders	4
2.2 Types of Pending Orders	4
2.3 Types of closing conditions	4
2.4 MyInvestia Dashboard	5
2.5 Investopedia Dashboard	6
2.6 MetaTrader4 Dashboard	6
3.1 Use Case Diagram	10
3.2 Class Diagram	11
A.1 Database Tables' Properties	21
A.2 Establishing a connection with our database	21
B.1 Pydantic in action: Data validation	23
B.2 Examples of SQLAlchemy : SELECT query & UPDATE	24
B.3 fastAPI: precise & concise code	24
B.4 Authorization Router: JWT & Hashing	25
B.5 fastFOREX API, the logic behind it	25

Chapter 1

Introduction

1 Context

Financial markets play a pivotal role in the global economic landscape, serving as the bedrock of our interconnected financial system. These markets encompass a diverse array of instruments, such as stocks, bonds, commodities, and currencies, where participants engage in the buying and selling of financial assets. The significance of financial markets extends far beyond the realm of numbers and charts, as they are integral to the functioning of economies worldwide.[1]

The increasing globalization of financial markets has further underscored their importance. With advancements in technology, information dissemination has become instantaneous, allowing investors from different corners of the world to participate in markets seamlessly. This global inclusion has both benefits and challenges, as demonstrated by the rapid transmission of economic shocks across borders, highlighting the need for effective regulatory frameworks.

Now, shifting focus to Tunisia, our country maintains a financial system that includes the Bourse des Valeurs Mobilières de Tunis (BVMT), a stock exchange facilitating the trading of securities. The Tunisian Dinar (TND) is the national currency, and while the country is not completely closed off, there are currency controls and restrictions in place. These measures include the prohibition of importing/exporting TND and limits on the possession of foreign currency by individuals, aiming to manage foreign exchange reserves and stabilize the national currency. [2]

2 Motivation

Such laws made it difficult for the average citizen to comprehend the intricate workings of the aforementioned financial markets, particularly the dynamic realm

of currency trading ¹. Even business and economics students only indulge in the theoretical side of the equation, never really getting headfirst into action. In response, this initiative aims to narrow this knowledge gap through the development of an API-based solution for a simple, educational trading platform. The underlying concept is straightforward yet powerful: to simplify and emulate a real market environment, instill a sense of competition, and provide users with the opportunity to engage in a risk-free simulated trading, thereby gaining practical insights into the complexities of the actual financial landscape. This hands-on approach aims to demystify the intricacies of currency markets.

¹Also known as forex or FX, currency markets see the buying and selling of global currencies 24 hours a day. It is the world's most traded market.

Chapter 2

Methodology

1 Research

As an initial step of my project, I undertook a comprehensive study to grasp the fundamentals of currency trading. The key insights obtained are as follows [3]

- **FX Traders and Trading Basics:**

- FX traders aim to profit from fluctuations in exchange rates between currencies by speculating on whether one currency will go up or down in value compared to another.
- Currencies are traded in pairs, represented by two abbreviations together, usually separated by a forward slash. For example, EURUSD represents trading Euros (base) against US dollars (quote).
- Major currency pairs include EURUSD, USDJPY, GBPUSD, USDCHF, USDCAD, AUDUSD, and NZDUSD.
- Minor/Crosses are currency pairs that don't include the U.S. dollar.
- Exotics involve a major currency paired with that of an emerging economy.

- **Terms and Measurements:**

- **Pip:** Percentage in point (0.0001). For example, if USDCAD goes from 1.41065 to 1.41075, it increases by 1 pip.
- **Lot:** A lot is often used in forex to quantify currency units. A standard lot equals 100,000 units of the base currency.
- **Bid and Ask:** The bid is the maximum a buyer on the forex market is willing to pay for a currency pair. The ask is the minimum a seller is willing to accept. The spread is calculated as Ask price – Bid price.

Market Buy Order	Market Sell Order
Immediately buy the currency pair at the available ask price	Immediately sell the currency pair at the available bid price.

Figure 2.1: Types of Market Orders

Limit Order		Stop Order	
Buy limit Order	Sell limit Order	Buy stop order	Sell stop order
Execute the order after the ask price falls down to a given value	Execute the order after the bid price goes up to a given value	Execute the order after the ask price goes up to a given value	Execute the order after the bid price falls down to a given value

Figure 2.2: Types of Pending Orders

- **Trading Strategies:**

- **Going Long:** Buying a currency pair with the intention of selling it later at a potentially higher price.
- **Going Short:** Selling a currency pair with the intention of buying it later at a potentially lower price.

- **Major Types of Orders:**

- **Market Orders:** Executed immediately at the best price available.
- **Pending Orders:** Executed after a condition has been met.
- **Closing Orders:** Set by the trader to close an existing open order after a certain condition has been met.

- **Account Balance and Equity:**

- **Current Balance:** The amount of money in the trading account. It does not include any profit or loss an open position might incur. Only updated after a trade is closed.
- **Equity:** Equity is calculated as the sum of the balance and the profit/loss from open positions. If there are no open positions, the balance is equal to equity.

Take profit (lock in the gains)		Stop loss (minimize losses)	
Buying context	Selling context	Buying context	Selling context
Close an open position after a price goes up to a given value.	Close an open position after a price goes down to a given value.	Close an open position after a price goes down to a given value.	Close an open position after a price goes up to a given value.

Figure 2.3: Types of closing conditions

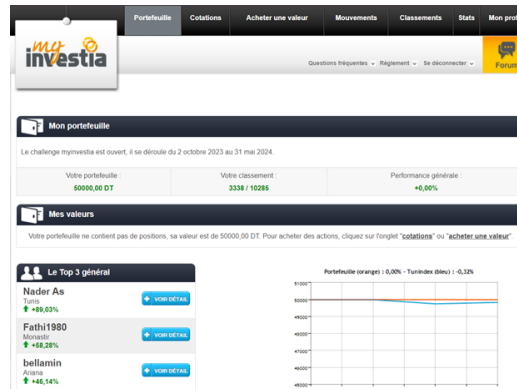


Figure 2.4: MyInvestia Dashboard

- **Leverage and Margin:**
 - **Leverage:** A loan provided by the broker (trading platform) acting as a multiplier, allowing you to control larger positions on the market with less of your own money. It is expressed as a ratio (1:alpha).
 - **Required Margin:** The amount of money you need to have in a leveraged account to keep a position open.
 - **Free Margin:** The money in a trading account available for trading. Free Margin is calculated as Equity minus the Required Margin of open positions.
- **Stop Out:** The forced closing of the worst open position after the Required Margin becomes more than the available equity.

2 Benchmarking

Benchmarking involves comparing our proposed API with existing solutions to evaluate its performance and features. Here, we explore three benchmarking platforms:

2.1 MyInvestia:

MyInvestia [4] is a risk-free online stock market simulation game organized by the Bourse de Tunis. Open to everyone and free of charge, it provides a unique opportunity to experience stock market investment using a virtual capital of 50,000 dinars. Participants build a portfolio of stocks from companies listed on the Bourse de Tunis, valued at real market prices. At the end of the challenge, the top six

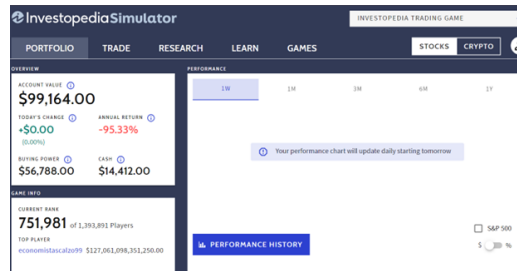


Figure 2.5: Investopedia Dashboard



Figure 2.6: MetaTrader4 Dashboard

portfolios have a chance to win cash prizes, with amounts ranging from 500 to 3,000 DT.

Despite being an excellent initiative from BVMT, this simulation is still limited and closed on our own economy because only the stocks of companies registered in that stock exchange are tradable.

2.2 Investopedia Simulator Game:

The Investopedia Simulator [5] is crafted to assist individuals in grasping the intricacies of purchasing and selling securities within a safe and risk-free setting. This educational tool mimics the processes investors go through when executing orders through a broker. It offers live monitoring of simulated investment value fluctuations, allowing users to observe the performance of their investments in real time. While this is a global scale simulation game, it is also only limited to stocks, options, and cryptocurrencies. As of today, they do not include the option of exchanging foreign currencies yet.

2.3 MetaTrader4 Demo Account:

A MetaTrader4 (MT4) [6] demo account is a simulated trading account provided by brokers to users who want to practice trading in a risk-free environment. MT4 is a popular trading platform widely used in the forex market. The demo account on MetaTrader4 allows traders, especially beginners, to familiarize themselves with

the platform’s features, test trading strategies, and gain hands-on experience in executing trades without using real money.

While heavily inspired by MetaTrader4, our API is custom-designed to operate within a meticulously regulated and controlled framework. Specifically, we contemplate introducing the TND (Tunisian Dinar) into the trading experience, limiting the choice to just 3 currency pairs. This deliberate choice is guided by our commitment to prioritize the educational aspect, creating a hypothetical scenario that encourages users to explore the potential impact of incorporating the TND into their trading strategies.

3 Rules Design

1. In order to be able to trade, a user has to create an account and login.
2. Once authenticated, each trader has access to a leveraged account with the following initial properties:
 - Balance= 5,000,000 TND.
 - Profit/Loss= 0.
 - Equity= 5,000,000 TND.
 - Leverage= 1:50
 - Required Margin= 0.
 - Free Margin= 5,000,000 TND.
 - Performance% = 0.
 - Status= “Active”
3. A trader can, at any time, view the current stats of their profile, which will be updated every minute depending on the exchange rates fluctuations. ¹
4. A trader can view a time-series of the exchange rates of the following currencies “TND/EUR”, “EUR/USD”, and “USD/CAD”.
5. A trader can view the current prices of “TND/EUR”, “EUR/USD”, and “USD/CAD”. These values are updated every time they call the endpoint.
6. A trader can create an order with the following structure:
 - Type= “market”, “limit” or “stop”
 - Objective= Either “Buy” or “Sell”.

¹This feature could not be implemented. See 3.8 Future Enhancements and Recommendations for more details

- Currency Pair= “TND/EUR” or “EUR/USD” or “USD/CAD”
 - Trigger Price = Current Market Price if it’s a market order, otherwise their Chosen Value.
 - Quantity in lots= Chosen value
 - Take profit= default None, otherwise Chosen Value
 - Stop loss= default None, otherwise Chosen Value
 - Status= Either “Open” or “Closed”
7. A trader can close an open trade.
 8. A trader can view their open and closed orders.
 9. A trader can change the “Stop Loss” and “Take Profit” properties of their open orders.
 10. A trader can view the stats of other players.
 11. A trader can delete their profile. This action also deletes their orders.

Chapter 3

Implementation

The code was heavily inspired by a fastAPI tutorial [7], with a few tweaks of the logic and the properties to fit into the criteria we set. But before getting into the coding ¹, a few things had to be set first - namely the architecture, and the technologies' choice.

1 Architecture

1.1 Use Case Diagram

This Use Case Diagram, shown in figure 3.1, illustrates the various interactions between different actors and the system, showcasing the primary functionalities and scenarios within the Forex Trading Simulation.

1.2 Class Diagram

This Class Diagram, shown in figure 3.2 provides a visual representation of the structure and relationships among the key classes within the Forex Trading Simulation system. It outlines the essential entities, their attributes, and the associations between classes, offering a comprehensive overview of the system's internal organization and data flow.

2 RDBMS Choice

PostgreSQL is a powerful open-source relational database management system (RDBMS). It serves as the backend database to persistently store and retrieve data.

¹For code snippets and brief explanations, see Appendix

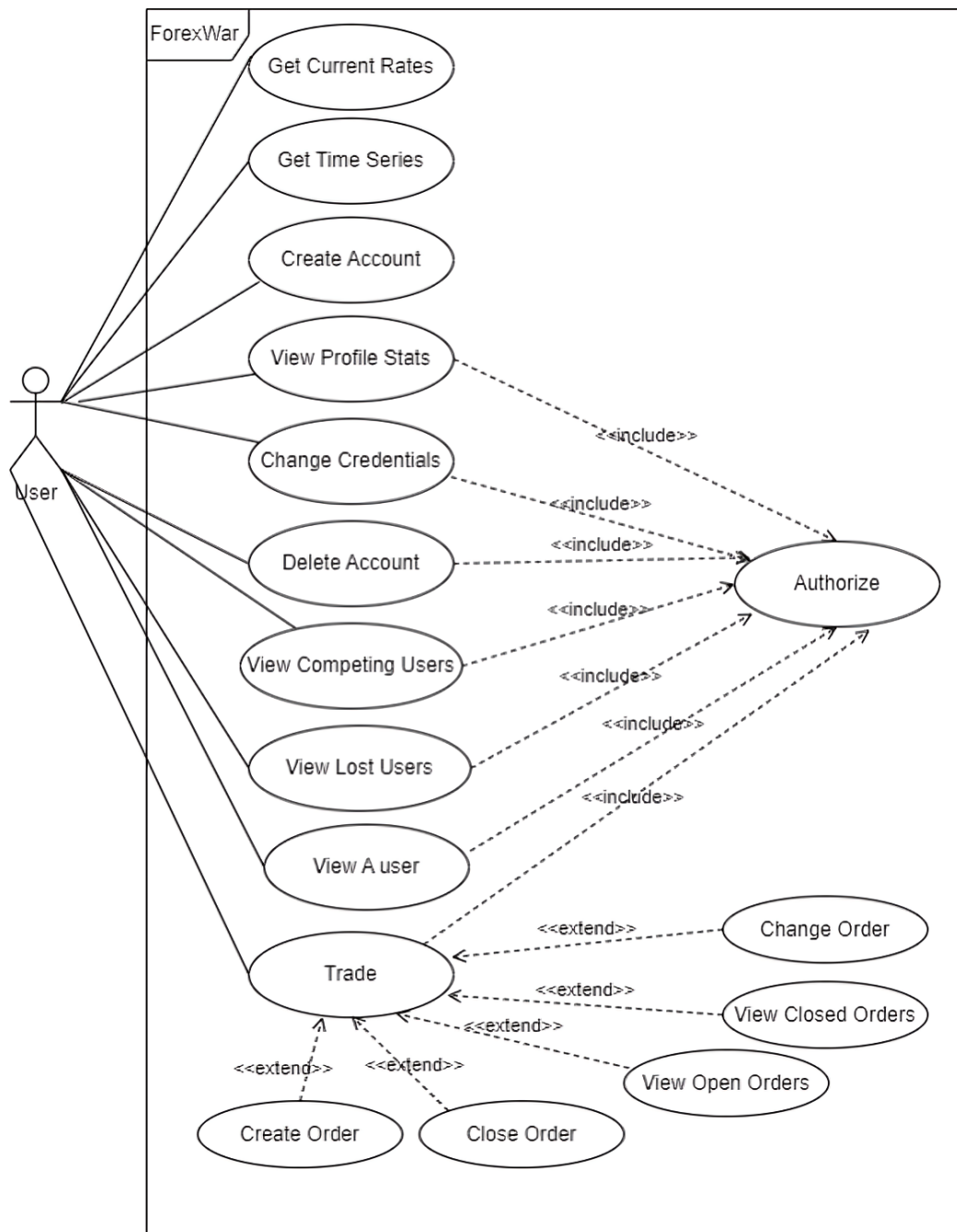


Figure 3.1: Use Case Diagram

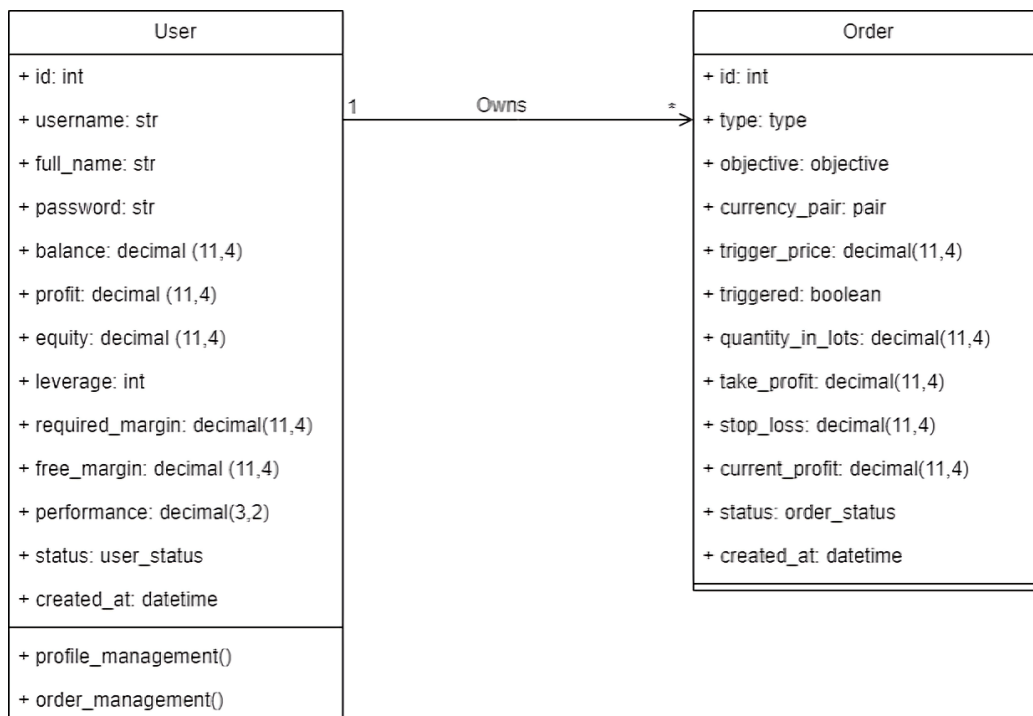


Figure 3.2: Class Diagram

3 Main Tools & Technologies Used

3.1 Visual Studio Code

Visual Studio Code significantly enhanced our project development with its intuitive interface, extensive Python language support, integrated terminal, Git integration, and a rich ecosystem of extensions. Visual Studio Code streamlined our development workflow.

3.2 FastAPI

This web development framework was chosen for this project mainly because of three reasons:

1. **Fast Development with Python:** FastAPI is built on top of Starlette and Pydantic ², leveraging the power of Python. This allows for rapid development with concise, easy-to-understand code. Python's simplicity and readability makes it an excellent choice for projects with complex functionalities.
2. **Automatic API Documentation:** FastAPI automatically generates interactive API documentation using Swagger UI and ReDoc. This feature is invaluable for developers and users alike, as it provides a real-time, user-friendly interface to understand, test, and explore the API endpoints. It reduces the manual effort needed for documentation maintenance.
3. **Security Features:** FastAPI has built-in support for OAuth2 authentication ², which is crucial for securing the API. It simplifies the implementation of secure authentication mechanisms, such as OAuth2PasswordBearer, enhancing the overall security of the forex market simulation API.

3.3 SQLAlchemy

SQLAlchemy is a powerful SQL toolkit and Object-Relational Mapping (ORM) library for Python. It plays a crucial role in managing database interactions, serving the project in three main aspects:

1. **Creating a Session and Engine:** SQLAlchemy uses a concept called a "session" to manage our database (PostgreSQL) connections and transactions.
2. **Defining Database Models:** SQLAlchemy allows us to define data models using Python classes. These models represent the structure of our database

²explained later

tables, with each class attribute typically corresponding to a column in the table.

3. **CRUD Operations:** SQLAlchemy simplifies CRUD (Create, Read, Update, Delete) operations.

3.4 Pydantic library:

This library was used to validate the request/response bodies. For each endpoint used in our routers, we should expect an exact request body, and return an exact response. Failure to provide the expected request will result in an exception error. These validations have been set in `schemas.py`.

4 Initiation

Version Control: GitHub played a pivotal role in our version control strategy. The process began by creating a remote repository titled `ForexWar_TN`. We proceeded to clone it into our local machine, establishing a virtual environment (`venv`), and specifying project requirements in the `requirements.txt` file. Subsequently, each set of changes was committed³ and pushed⁴ to the remote repository. This iterative cycle continued, with commits marking significant development milestones.

5 Security Implementation

Security [8] was arguably the most challenging phase. After setting up our database and the users' table, it was time to implement a secure authorization system. This was accomplished through the `auth.py` router and the `oauth2.py` file.

5.1 Authorization Router

The endpoint function `login` takes two parameters:

- **user_credentials:** An instance of `OAuth2PasswordRequestForm`. This represents the form data for the username and password sent in the request.
- **db:** An instance of the SQLAlchemy Session obtained through the `database.get_db` dependency. It represents the database session.

³Using the command: `git commit -m "commit message"`

⁴Using the command: `git push origin main`

We then perform a database query by selecting a matching email. If none is found, we return an exception. If we find a corresponding email but the passwords ⁵ are not the same, we also return an exception. After that, we create a token using a function from the `oauth2.py` file and return it.

5.2 `oauth2.py` file

This file takes care of creating and verifying the access tokens.

Creation of the access token: This function takes data as an argument (in our context, it is the `user.id`). We then copy this data and encode it using the `jwt.encode` function (which takes the copy, the `secret_key`, and the algorithm of our choice). It finally returns the `encoded_jwt`⁶. Later on, while testing the API, we used a token tag to avoid copying and pasting it every time to access the routes.

Verification of the access token: This was used every time the user wanted to access a secure router. If this function returns an exception (i.e., not authorized), the user is not allowed to use the route we used this function on.

`get_current_user()` is used later on as an argument for our functions. It takes the token and the session, verifies the token using `verify_access_token()`, looks for the user with the available `token.id`, and returns it. `verify_access_token()` uses the `jwt.decode()` function to check the payload stored in that token. If no `id` is found in it, it returns a `credentials_exception`, and the user is forbidden from accessing that endpoint.

6 Profile Router

This phase was very straightforward.

- A create endpoint was used for the user to allow him to create a new profile. His data is then stored in the database. Duplicate emails or usernames are not allowed, and an error is returned in case that happens.
- Several get endpoints are available for the user once authorized. He can check his own profile and check other users' profiles.
- A put request is also available where he can change his username, email, full name, or password.
- A delete request allows the user to delete his account from the database. Doing so also deletes all of the trades he created in the orders' table (`on_delete="CASCADE"`).

⁵hashed, using our utilities class, which uses the `passlib.context` library to encrypt the password and compare their hash codes.

⁶Note: This token is then stored in the client (browser)

7 Trading Router

Only authorized users can trade. We utilized the `oauth2.get_current_user` dependency for that purpose.

- **Creation Request of an Order:** In financial terms, this is referred to as "opening a position." Several checkpoints are in place to verify that the request body complies with the specified order types. For example, a market order cannot have a trigger price, while a pending order must contain a trigger price. Depending on the current currency pair price ⁷, the trigger price has to be in a certain range. Failure to comply with these conditions raises an exception.
- **Put Request:** In financial terms, this is known as "closing a position." It occurs when the user decides to realize the gains/losses from that position. Users can only close their own orders.
- **Another Put Request:** This request is used to change the stop-loss or take-profit values. Similarly, this order must be owned by the user, and it has to be open. Users cannot change closed orders.
- **2 Get Requests:** These are implemented to allow the user to view their own orders, both closed and open.

8 Currencies' Router

This router is available for all users (authorized or not). It allows them to check the current rates of TNDEUR, EURUSD, and USDCAD, which they can trade. It also has a second get endpoint, allowing them to get the historical data of a current currency pair for the last 14 days. This was achievable thanks to fastFOREX.

8.1 fastFOREX API integration :

Our project's integration of the fastFOREX API [9] is underpinned by its exceptional utility and robust features. Delivering precise foreign currency exchange rate data and real-time pricing for over 300 currencies, the API aligns seamlessly with our real market rates objective.

On top of this huge benefit, this choice was vastly due to the following factors:

- They offer a 7-day free trial API key for each new sign-up. The project submission and presentation were set to be in less than a week, so it was an opportunity to be taken. In this trial period, we get a generous limit (1 million calls) allowing us to test the results as many times as we want.

⁷see: next Currencies Router section

- Ease of integration: the code to integrate is fairly simple. We just need to install and import the requests package. And then call the desired endpoints inside our own routes.

8.2 Get_current_exchange_rates:

This calls the following endpoint 3 times⁸ :

```
api.fastforex.io/fetch-multi?from=BASE&to=TARGET&api_key=APIKEY
```

And performs a set of operations to determine the current price of TNDEUR, EURUSD, and USDCAD, which are stored globally.

8.3 Get_time_series:

The user inputs the “base” currency and the “target” currency in the request body, to get a time-series of the exchange rates in the last 14 days (the max duration offered by the free trial in fastFOREX). The following endpoint⁹ is used and its output is the response of our route:

```
api.fastforex.io/time-series?from=BASE&to=TARGET&start=START&end=TODAY&api_key=APIKEY
```

9 Testing

Insomnia is a robust and user-friendly REST API client that played a pivotal role in the continuous testing of our project. Serving as a comprehensive tool, Insomnia allowed for the seamless creation and execution of HTTP requests, enabling thorough testing of our API endpoints. To enhance efficiency, we implemented a base environment URL variable within Insomnia, eliminating the need to manually type the local host every time

Additionally, to streamline access to secure endpoints requiring authentication, an automatic authentication bearer token was configured. This automated approach spared the need for entering the lengthy token string manually for each request, simplifying the testing process and ensuring a more efficient and error-free exploration of our API’s functionalities.

⁸BASE: the base currency/ TARGET: the quote currency/ APIKEY: the API key we received upon signup

⁹START: 14 days ago / TODAY: the current day

10 Future Enhancements and Recommendations

- **Profit calculation:** While the current project is a work in progress, a significant feature yet to be implemented is profit calculation. Despite the simplicity of the financial formula (buy orders: $(currentprice - buyprice) * quantity$, sell orders: $(sellprice - currentprice) * quantity$), implementing backend logic to continuously update profit columns for users' and orders' tables has proven challenging. This crucial functionality remains a priority for future development.
- **Database Migration with Alembic:** Alembic provides a powerful and flexible solution for database schema versioning and it works well with SQLAlchemy. Implementing Alembic can streamline the process of managing database changes, allowing for smooth transitions and updates without manual intervention
- **Front-end** Enhance the user experience by developing a front-end that seamlessly integrates with our API. A well-designed front-end will not only provide a more user-friendly interface but also facilitate user interaction with the core functionalities of the application, making it more accessible and engaging
- **Deployment** Whether it's deploying on cloud services or setting up a dedicated server, a well-thought-out deployment strategy is crucial for making our API accessible to users.
- **More testing** There was no automated testing implementations in this API. All of the testing was done manually, and this raises a great possibility of exception mishandling.

Chapter 4

Conclusion and Perspectives

The ForexWarTN API project is a crucial step in making currency trading accessible and educational, especially in Tunisia. Our initiative simplifies the complexities of financial markets, providing a user-friendly simulation for practical learning. Thorough research and benchmarking informed the rules design, resulting in a streamlined forex trading simulation.

Implementation involved a robust architecture utilizing PostgreSQL as our RDBMS, and FastAPI & SQLAlchemy as our web frameworks. The integration of fastAPI has proven very useful, despite the 7-day trial period. And finally, for testing, Insomnia was used.

Future plans include implementing profit calculation, exploring database migration with Alembic, integrating a front-end, planning deployment, and continuous testing for reliability. This project is a milestone in empowering users, particularly students in Tunisia, to gain hands-on experience in navigating financial markets. Ongoing development aims to further enhance accessibility and educational value.

Bibliography

- [1] forex.com. *Introduction to Financial Markets*. URL: www.forex.com/ie/trading-academy/courses/introduction-to-financial-markets.
- [2] U.S. Embassy in Tunisia. *Currency Restrictions in Tunisia*. URL: <https://tn.usembassy.gov/u-s-citizen-services/security-and-travel-information/currency-restrictions-in-tunisia>.
- [3] Trading.com. *What is Forex*. URL: <https://www.trading.com/us/learn/tutorials/what-is-forex>.
- [4] BMVT. *MyInvestia*. URL: <https://www.myinvestia.com/>.
- [5] Investopedia. *Investopedia Simulator*. URL: <https://www.investopedia.com/simulator/portfolio>.
- [6] *MetaTrader4*. URL: <https://www.metatrader4.com/en>.
- [7] Sanjeev-Thiyagarajan. *Python API Development - Comprehensive Course for Beginners*. URL: <https://github.com/Sanjeev-Thiyagarajan/fastapi-course>.
- [8] Tiangolo. *FastAPI OAuth2 with JWT Tutorial*. URL: <https://fastapi.tiangolo.com/tutorial/security/oauth2-jwt/>.
- [9] *FastFOREX API Documentation*. URL: <https://fastforex.readme.io/reference/introduction>.

Appendix A

Database

The screenshot displays two side-by-side windows for configuring database tables. The left window is for the 'users' table, and the right window is for the 'orders' table. Both windows show a 'Columns' tab with a table of column properties.

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
id	integer			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	nextval('us
username	character varying			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
full_name	character varying			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
email	character varying			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
password	character varying			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
balance	numeric			<input type="checkbox"/>	<input type="checkbox"/>	5000000.0
profit	numeric			<input type="checkbox"/>	<input type="checkbox"/>	0.0000
equity	numeric			<input type="checkbox"/>	<input type="checkbox"/>	5000000.0
leverage	integer			<input type="checkbox"/>	<input type="checkbox"/>	50
required_margin	numeric			<input type="checkbox"/>	<input type="checkbox"/>	0.0000
free_margin	numeric			<input type="checkbox"/>	<input type="checkbox"/>	5000000.0
performance	numeric			<input type="checkbox"/>	<input type="checkbox"/>	0.0000
status	character varying			<input type="checkbox"/>	<input type="checkbox"/>	'competing
created_at	timestamp with tim...			<input checked="" type="checkbox"/>	<input type="checkbox"/>	now()

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
id	integer			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	nextval('or
type	character varying			<input checked="" type="checkbox"/>	<input type="checkbox"/>	'market':cl
objective	character varying			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
currency_pair	character varying			<input checked="" type="checkbox"/>	<input type="checkbox"/>	
trigger_price	numeric	11	4	<input type="checkbox"/>	<input type="checkbox"/>	NULL:null
triggered	boolean			<input type="checkbox"/>	<input type="checkbox"/>	false
quantity_inLots	numeric	11	4	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
take_profit	numeric	11	4	<input type="checkbox"/>	<input type="checkbox"/>	NULL:null
stop_loss	numeric	11	4	<input type="checkbox"/>	<input type="checkbox"/>	NULL:null
current_profit	numeric	11	4	<input type="checkbox"/>	<input type="checkbox"/>	0.0000
status	character varying			<input type="checkbox"/>	<input type="checkbox"/>	'open':cha
created_at	timestamp with tim...			<input checked="" type="checkbox"/>	<input type="checkbox"/>	now()
owner_id	integer			<input checked="" type="checkbox"/>	<input type="checkbox"/>	

Figure A.1: Database Tables' Properties

```

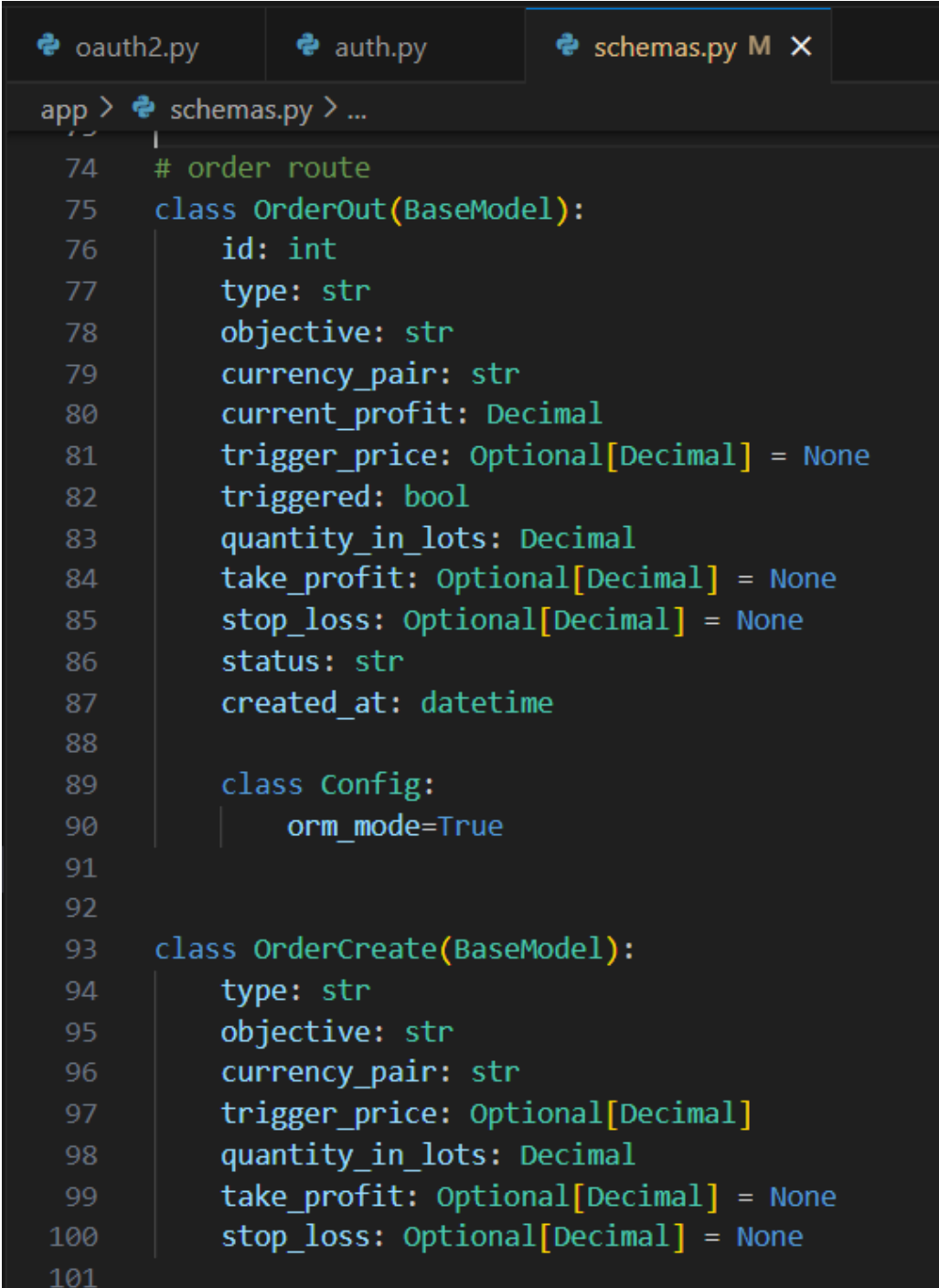
database.py X
app > database.py > ...
1  from sqlalchemy import create_engine
2  from sqlalchemy.ext.declarative import declarative_base
3  from sqlalchemy.orm import sessionmaker
4
5  SQLALCHEMY_DATABASE_URL = "postgresql://postgres:root@localhost/forexwar_tn"
6
7  engine = create_engine(SQLALCHEMY_DATABASE_URL)
8  SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
9
10 Base = declarative_base()
11
12 # Dependency
13 def get_db():
14     db = SessionLocal()
15     try:
16         yield db
17     finally:
18         db.close()
19
20 print("Database connection successful.")

```

Figure A.2: Establishing a connection with our database

Appendix B

Code Snippets



```
oauth2.py  auth.py  schemas.py M X
app > schemas.py > ...
74 # order route
75 class OrderOut(BaseModel):
76     id: int
77     type: str
78     objective: str
79     currency_pair: str
80     current_profit: Decimal
81     trigger_price: Optional[Decimal] = None
82     triggered: bool
83     quantity_in_lots: Decimal
84     take_profit: Optional[Decimal] = None
85     stop_loss: Optional[Decimal] = None
86     status: str
87     created_at: datetime
88
89     class Config:
90         orm_mode=True
91
92
93 class OrderCreate(BaseModel):
94     type: str
95     objective: str
96     currency_pair: str
97     trigger_price: Optional[Decimal]
98     quantity_in_lots: Decimal
99     take_profit: Optional[Decimal] = None
100    stop_loss: Optional[Decimal] = None
101
```

Figure B.1: Pydantic in action: Data validation

```

@router.put("/orders_closing/{order_id}", response_model=schemas.OrderOut)
def close_order(order_id: int, db: Session = Depends(get_db), current_user: models.User = Depends(oauth2.get_current_user)):
    # Retrieve the order from the database
    existing_order = db.query(models.Order).filter(models.Order.id == order_id).first()

    # Check if the order exists
    if not existing_order:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Order not found")

    # Check if the current user owns the order
    if existing_order.owner_id != current_user.id:
        raise HTTPException(status_code=status.HTTP_403_FORBIDDEN, detail="You are not the owner of this order")

    # set status to closed
    existing_order.status = "closed"

    # Commit the changes to the database
    db.commit()
    db.refresh(existing_order)

    return existing_order

```

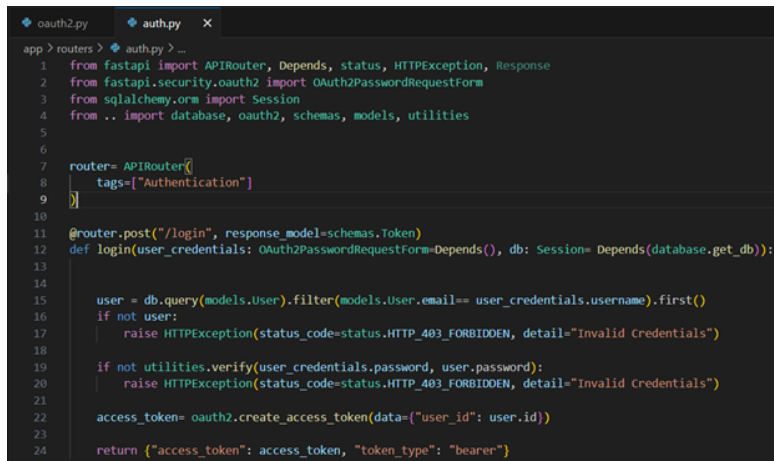
Figure B.2: Examples of SQLAlchemy : SELECT query & UPDATE

```

main.py M X
app > main.py > root
1  from fastapi import FastAPI
2  from .database import engine
3  from . import models
4
5  from .routers import user, auth, order, currencies
6
7  models.Base.metadata.create_all(bind=engine)
8
9  app = FastAPI()
10 app.include_router(currencies.router)
11 app.include_router(auth.router)
12 app.include_router(user.router)
13 app.include_router(order.router)
14
15
16 @app.get("/")
17 async def root():
18     return {"message": "Hello World"}

```

Figure B.3: fastAPI: precise & concise code

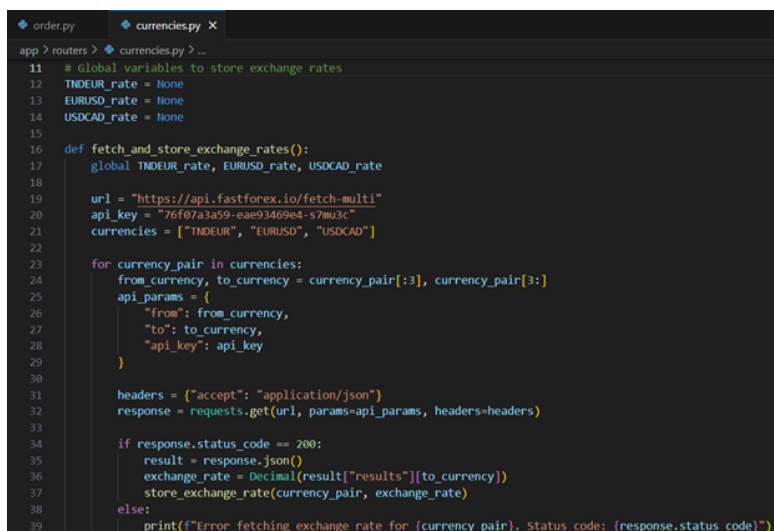


```

1 from fastapi import APIRouter, Depends, status, HTTPException, Response
2 from fastapi.security.oauth2 import OAuth2PasswordRequestForm
3 from sqlalchemy.orm import Session
4 from .. import database, oauth2, schemas, models, utilities
5
6
7 router = APIRouter(
8     tags=["Authentication"]
9 )
10
11 @router.post("/login", response_model=schemas.Token)
12 def login(user_credentials: OAuth2PasswordRequestForm=Depends(), db: Session=Depends(database.get_db)):
13
14     user = db.query(models.User).filter(models.User.email== user_credentials.username).first()
15     if not user:
16         raise HTTPException(status_code=status.HTTP_403_FORBIDDEN, detail="Invalid Credentials")
17
18     if not utilities.verify(user_credentials.password, user.password):
19         raise HTTPException(status_code=status.HTTP_403_FORBIDDEN, detail="Invalid Credentials")
20
21     access_token = oauth2.create_access_token(data={"user_id": user.id})
22
23     return {"access_token": access_token, "token_type": "bearer"}

```

Figure B.4: Authorization Router: JWT & Hashing



```

11 # Global variables to store exchange rates
12 TND EUR_rate = None
13 EUR USD_rate = None
14 USD CAD_rate = None
15
16 def fetch_and_store_exchange_rates():
17     global TND EUR_rate, EUR USD_rate, USD CAD_rate
18
19     url = "https://api.fastforex.io/fetch-multi"
20     api_key = "76f07a3a59-eae93469e4-s7mu3c"
21     currencies = ["TND EUR", "EUR USD", "USD CAD"]
22
23     for currency_pair in currencies:
24         from_currency, to_currency = currency_pair[:3], currency_pair[3:]
25         api_params = {
26             "from": from_currency,
27             "to": to_currency,
28             "api_key": api_key
29         }
30
31         headers = {"accept": "application/json"}
32         response = requests.get(url, params=api_params, headers=headers)
33
34         if response.status_code == 200:
35             result = response.json()
36             exchange_rate = Decimal(result["results"][to_currency])
37             store_exchange_rate(currency_pair, exchange_rate)
38         else:
39             print(f"Error fetching exchange rate for {currency_pair}. Status code: {response.status_code}")

```

Figure B.5: fastFOREX API, the logic behind it