

A  
PROJECT DOCUMENTATION  
ON  
**Yahoo Stock Price Notifier**



**SUBMITTED BY**  
Anish Shilpakar

**SUBMITTED TO**  
Coderush Nepal

April 23, 2023

# **TABLE OF CONTENTS**

<b>INTRODUCTION</b>	<b>3</b>
<b>TOOLS AND TECHNOLOGIES USED</b>	<b>4</b>
<b>METHODOLOGY</b>	<b>5</b>
<b>CONCLUSION</b>	<b>19</b>

# INTRODUCTION

This is the challenge project 1 in the Coderush Data Engineering Apprenticeship program. Here in this project I have used various tools and technologies to complete the given project tasks as mentioned in the objectives section. This is a simple web app project developed using Flask framework of Python. Here the frontend of web application presents user with a input form, where user can enter the stock ticker symbol, set stock price threshold, set frequency of notification and notification mode. Then based on user's settings user will get notified using email/SMS in the email address and phone number they have provided. To complete the project objectives, I have used various python libraries and APIs which are mentioned in Tools and Technologies section.

## PROJECT OBJECTIVES

- To build a web application that allows users to receive notifications when the price of a given stock reaches certain threshold
- To allow users to specify the frequency at which the stock price should be checked (every minute, hour, day)
- To allow users to specify the type of notification they would like to receive (email, text message)

## SCOPE AND APPLICATIONS

The project can have following scope and applications

1. **Personal Investment Management:** Individual investors can use this app to track the performance of their investments in the stock market. They can set specific price points that will trigger notifications, allowing them to make timely investment decisions.
2. **Trading:** Traders can use this app to monitor the prices of various stocks and take advantage of trading opportunities as they arise. The app can provide real-time alerts to help traders make informed decisions quickly.
3. **Investment Analysis:** Investors and analysts can use the app to monitor stock prices and analyze trends in the market. The app can provide real-time data on stock prices, trading volumes, and other metrics, enabling users to make informed investment decisions.
4. **News and Information:** The app can also provide news and information related to the stocks being monitored. This can include news articles, earnings reports, and other financial data that can help users make informed decisions about their investments.
5. **Risk Management:** The app can be used to manage risk by setting alerts for specific price points. If a stock price falls below a certain threshold, the app can notify users, allowing them to take action to mitigate their losses.

# TOOLS AND TECHNOLOGIES USED

## 1. Python

Python is a high-level, interpreted, and general-purpose programming language. Python offers various key features like simplicity, readability, versatility, etc. making it suitable for a wide range of applications like web development, scientific computing, data engineering, data analysis, data processing, machine learning and more. Also, Python is also an object-oriented programming language, meaning that it is built around the concept of objects and classes. This makes it easy to create complex and modular programs, as well as to reuse code and make modifications to existing code.

## 2. Flask

Flask is a lightweight and popular web framework for building web applications in Python. It is based on the Werkzeug toolkit and the Jinja2 template engine, and provides a simple and flexible way to create web applications. Flask is known for its simplicity, minimalism, and ease of use, making it a popular choice for beginners and experienced developers alike. Flask supports a wide range of extensions, making it highly customizable and adaptable to different use cases. Flask was used to develop the backend of the web application.

## 3. Yfinance

Yfinance is a python library available in pip that offers a threaded and Pythonic way to download market data from Yahoo finance. In this project, I have used yfinance to get the latest US market stock data.

## 4. Twilio

Twilio is a cloud communication platform that provides developers with a set of APIs for building SMS, voice, and messaging applications. With Twilio, developers can easily integrate communication capabilities into their software applications, websites, and mobile apps. Twilio's APIs are available in several programming languages, including Python, Ruby, PHP, and Java, among others. I have used Twilio to send text message to mobile number.

## 5. Schedule library

The schedule library in Python is a lightweight library that provides an easy way to schedule tasks to run at specified intervals. It allows developers to define a set of tasks as functions and schedule them to run periodically, either at specific times or at fixed intervals. I have used this library to schedule the notifications for users.

## 6. SQLite3 database

SQLite is a lightweight, file-based relational database management system (RDBMS) that is widely used for embedded systems and small-scale applications. It is serverless and self-contained, meaning that it requires no separate server process and all database operations are performed within the same process as the application using the database. SQLite uses SQL (Structured Query Language) to manage data and supports standard relational database features such as transactions, triggers, and indexes. SQLite databases are stored in a single file, making them easy to transport and backup. I have used this database for storing user's subscription input.

# METHODOLOGY

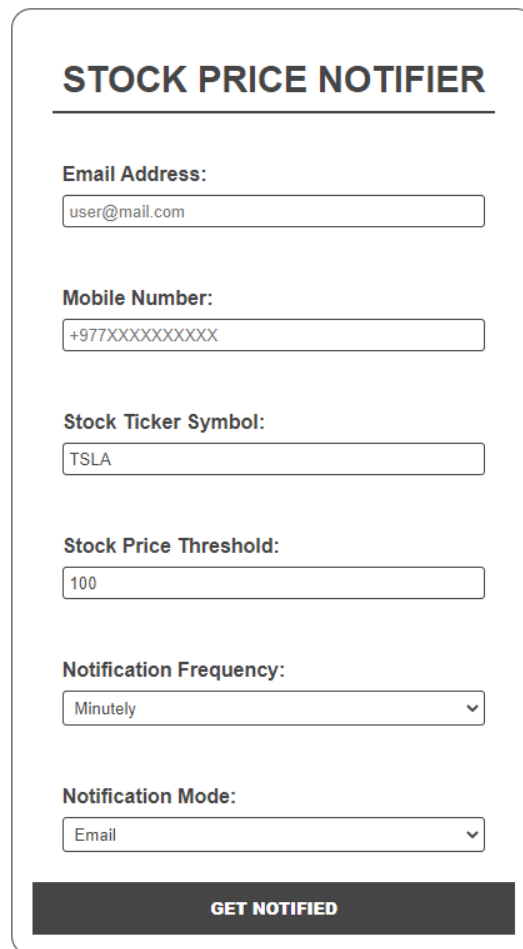
## 1. WORKING FLOW OF THE PROJECT

The working flow of “Yahoo Stock Price Notifier” includes following steps:

1. User input from HTML form
2. Input validation
3. Add subscriber to database
4. Fetch data from Yahoo Finance
5. Schedule Notifications
6. Send Notifications
7. Redirect to subscription success page

### 1. User input from HTML form:

In this project the frontend of web app is developed using basic HTML, CSS and backend is developed using Flask framework. The homepage consists of an input form which is used to get the details as input from user. Using this form, email address, mobile number, stock ticker symbol, stock price threshold, notification frequency and notification mode is taken as input from user. As the form fields have been set as required, so form can't be submitted until user has filled all the required fields. The input is then validated and stored in database as a new entry. The input form is shown below:



The image shows a web form titled "STOCK PRICE NOTIFIER". It contains several input fields and two dropdown menus. The fields are labeled "Email Address:", "Mobile Number:", "Stock Ticker Symbol:", "Stock Price Threshold:", "Notification Frequency:", and "Notification Mode:". The "Email Address" field contains "user@mail.com", "Mobile Number" contains "+977XXXXXXXXXX", "Stock Ticker Symbol" contains "TSLA", and "Stock Price Threshold" contains "100". The "Notification Frequency" dropdown is set to "Minutely" and the "Notification Mode" dropdown is set to "Email". At the bottom of the form is a dark button labeled "GET NOTIFIED".

Figure 1: User Input Form

## 2. Input Validation:

After taking the user input, the input needs to be validation before further processing as it helps to prevent further errors during course of project.

Firstly, email and phone number were validated using regex to check if they were valid email and number or not. This validation prevents error when sending notifications using email/text message. The code for email and phone number validation is shown below

```
# To validate if the email exists or not
def validate_email(email):
    # Email regex pattern
    email_pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

    # Check if email matches pattern
    if re.match(email_pattern, email):
        return True
    else:
        return False

# To validate if the phone number exists or not
def validate_number(number):
    # Phone number regex pattern
    phone_number_pattern = r'^+\d{1,3}\d{10}$'

    # Check if phone number matches pattern
    if re.match(phone_number_pattern, number):
        return True
    else:
        return False
```

Then the stock ticker symbol was validated. It was checked if the stock ticker symbol entered was user actually existed or not, if it didn't exist appropriate error message was shown to user. The code For stock ticker symbol validation is shown below

```
# function to check if the ticker exists or not
def check_ticker_exists(symbol):
    try:
        msft = yf.Ticker(symbol)
        data = msft.info
        return True
    except:
        return False
```

Then the stock price threshold was validated. Here I simply checked if the entered price threshold was a valid number or not. The code is shown below

```
# To validate if the price is valid number or not
def validate_price(price):
    try:
        price = float(price)
        return True
    except:
        return False
```

If error occurred during form validation, user will be displayed an error message as shown below with different messages for different errors.

The form is titled "STOCK PRICE NOTIFIER" and contains the following fields:

- Email Address:** A text input field containing "user@mail.com".
- Mobile Number:** A text input field containing "+977XXXXXXXXXX".
- Stock Ticker Symbol:** A text input field containing "TSLA".
- Stock Price Threshold:** A text input field containing "100".
- Notification Frequency:** A dropdown menu with "Minutely" selected.
- Notification Mode:** A dropdown menu with "Email" selected.
- GET NOTIFIED**: A dark button at the bottom of the form.

You Have Entered An Invalid Email Address. Please Enter A Valid Email Address In Format User@Mail.Com.

Figure 2: Form Input Validation Output

### 3. Add Subscriber to database:

After the user input from form has been validated, entry will be added to database for new subscriber. When adding the entry, (email, stock ticker symbol) or (phone number, stock ticker symbol) should be unique as I am considering one user won't add multiple subscription for same symbol using same email/phone number. In this project, I have connected the database with web application with to store the user's details for future use although it wasn't mentioned in task description. Also, using database helped me fetch the subscribers based on their frequency more easily.

The schema for database is shown below:

```
-- Schema for the subscribers table
DROP TABLE IF EXISTS subscribers;

CREATE TABLE subscribers(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    email TEXT NOT NULL,
    phone_number TEXT NOT NULL,
    symbol TEXT NOT NULL,
    threshold FLOAT NOT NULL,
    frequency TEXT NOT NULL,
    notification_mode TEXT NOT NULL,
    UNIQUE(email,symbol),
    UNIQUE(phone_number,symbol)
);
```

The following code was used to connect to the database and add entry for new subscriber:

```
# connecting to database
def get_db_connection():
    conn = sqlite3.connect('./database.db')
    conn.row_factory = sqlite3.Row
    return conn

# add new user to database
def
add_user(email,phone_number,symbol,threshold,frequency,notification_mode):
    conn = get_db_connection()
    conn.execute("INSERT INTO subscribers
(email,phone_number,symbol,threshold,frequency,notification_mode) VALUES
(?,?,?,?,?,?)",
(email,phone_number,symbol,threshold,frequency,notification_mode))
    conn.commit()
    conn.close()
    print("New Subscriber Added")
```



#### 4. Fetch data from Yahoo Finance (Data Collection):

In the task it was mentioned that we needed to use Yahoo Finance API to collect the required data, so I firstly used a Freemium Yahoo Finance API provided by RapidAPI, but this API only allowed 500 requests per month and it was slow. So later I used yfinance library available in PIP to collect the Yahoo finance data. When using yfinance, we need to specify the stock ticker symbol of the company we want, and then it will return all latest details about that stock. This was fast and more reliable, so I have used yfinance for data collection. After receiving all the data, I checked the response and then only used the required data from response. For this web app, only current price of stock is required as we will compare the current stock price with the price threshold set by user. So, I created a function called fetch\_data that takes stock\_ticker\_symbol as input and then returns the current stock price of that stock. The respective codes are shown below:

For collecting data using Yahoo Finance API

```
# function to fetch data from api (Optional)
def fetch_data_from_api(symbol="TSLA"):
    # API endpoint
    url = "https://apidojo-yahoo-finance-v1.p.rapidapi.com/stock/v3/get-
historical-data"
    # parameters for API
    region = "US"
    end_date = datetime.now().strftime("%Y-%m-%d")
    start_date = (datetime.now() - timedelta(days=30)).strftime("%Y-%m-%d")
    querystring = {"symbol": symbol, "region": region, "from": start_date,
"to": end_date}
    # API headers
    headers = {
        "X-RapidAPI-Key": RAPID_API_KEY,
        "X-RapidAPI-Host": RAPID_API_HOST
    }
    # API response
    response = requests.request("GET", url, headers=headers,
params=querystring)
    # parse API response to json format
    data = response.json()
    # create a dataframe from the json response
    df = pd.DataFrame(data["prices"])
    # to convert from unix timestamp to date
    df["date"] = pd.to_datetime(df["date"], unit="s").dt.date
    # only keep the columns we need for price
    df_price = df[['date', 'open', 'high', 'low', 'close', 'volume']]
    # print(df_price.head())
    return df_price
```

For collecting data using yfinance library

```
# function to fetch data using yfinance (I used this)
def fetch_data(symbol="TSLA"):
    msft = yf.Ticker(symbol)
    # getting ome price details from this response
    prevClosingPrice = msft.info['previousClose']
    openPrice = msft.info['open']
    dayLowPrice = msft.info['dayLow']
    dayHighPrice = msft.info['dayHigh']
    currentPrice = msft.info['currentPrice']
    volume = msft.info['volume']
    # get all stock info
    result = {
        "prevClosingPrice": prevClosingPrice,
        "openPrice": openPrice,
        "dayLowPrice": dayLowPrice,
        "dayHighPrice": dayHighPrice,
        "currentPrice": currentPrice,
        "volume": volume,
    }
    # print(result)
    # returning only the current price for further processing
    return currentPrice
```

## 5. Schedule Notifications:

To schedule notifications, schedule library was used. In web app, user has input to schedule the notifications either by minute, hour or day. The following function was used for scheduling notification

```
# for scheduling notification
def schedule_notification():
    schedule.every().minute.do(notify_minutely_subscriber)
    schedule.every(1).hour.do(notify_hourly_subscriber)
    schedule.every().day.at("08:00").do(notify_daily_subscriber)
    while True:
        schedule.run_pending()
        time.sleep(1)
```

Here schedule will check in every minute, hour and day (at time 8:00 AM) and then call the respective functions to notify the subscribers. The function to notify subscriber is shown below:

```
# function to notify subscribers based on their frequency
def notify_minutely_subscriber():
    print("Notifying minutely subscriber at", datetime.now())
```

```

minutely_subscribers = get_users("minutely")
if len(minutely_subscribers) > 0:
    notify_subscribers(minutely_subscribers)
    print(f"Notified {len(minutely_subscribers)} subscribers")
else:
    print("No minutely subscribers yet!")

```

Here, firstly the list of minutely subscribers was fetched from the database using `get_users()` function and then all the subscribers were sent the notification by calling `notify_subscribers()` function. The `get_users()` function will take frequency as input and then returns the list of subscribers based on their frequency. It is shown below:

```

# fetch all existing users from database
def get_users(subscribe_mode="minutely"):
    conn = get_db_connection()
    all_subscribers = conn.execute('SELECT * FROM subscribers ORDER BY
created DESC').fetchall()
    minutely_subscribers = conn.execute('SELECT * FROM subscribers WHERE
frequency="minute").fetchall()
    hourly_subscribers = conn.execute('SELECT * FROM subscribers WHERE
frequency="hour").fetchall()
    daily_subscribers = conn.execute('SELECT * FROM subscribers WHERE
frequency="day").fetchall()
    conn.close()
    print(f"There are {len(minutely_subscribers)} minutely subscribers
currently")
    print(f"There are {len(hourly_subscribers)} hourly subscribers
currently")
    print(f"There are {len(daily_subscribers)} daily subscribers
currently")
    if subscribe_mode == "minutely":
        return minutely_subscribers
    elif subscribe_mode == "hourly":
        return hourly_subscribers
    elif subscribe_mode == "daily":
        return daily_subscribers
    elif subscribe_mode == "all":
        return all_subscribers

```

After getting the list of subscribers, `notify_subscribers()` function is used to notify all the subscribers in the list. Before notifying, it is checked if the current price for the given stock is greater than the user's threshold value. And notification will be sent in user's preferred mode only if the condition is satisfied. To send notification `send_notification()` function is used which is explained in the next section. The following is the code for `notify_subscribers()`.

```
def notify_subscribers(subscribers):
    for subscriber in subscribers:
        currentPrice = fetch_data(subscriber['symbol'])
        if currentPrice > subscriber['threshold']:
            send_notification(subscriber['symbol'], subscriber['threshold'],
currentPrice, subscriber['email'], subscriber['phone_number'], subscriber['fre
quency'], notification_mode=subscriber['notification_mode'], notification_typ
e="price_update")
```

Also, to keep the scheduler running in background when the web app is running, I created another thread using threading library and then scheduling function runs in this separate thread. The following code was used to create the thread.

```
# Background thread for sending notifications
t1 = threading.Thread(target=schedule_notification, name="Schedule
Notifications")
t1.start()
```

## 6. Send Notifications:

The notification is sent to the user based on their notification mode. User gets option to choose either email or text message as their notification mode and then notification will be sent on respective channel using send\_notification() function. The code for send\_notification() function is shown below:

```
# function to send notification
def send_notification(symbol, threshold, currentPrice, subscriber_email,
subscriber_number, frequency, notification_mode="email", notification_type="pr
ice_update"):
    if notification_mode == "email":
        # send email for new entry
        send_mail(symbol, threshold, currentPrice, subscriber_email, frequency,
notification_type)
    elif notification_mode == "text-msg":
        # send message for new entry
        send_message(symbol, threshold, currentPrice, subscriber_number, freque
ncy, notification_type)
```

### A. Sending Email

To send email to subscribers, I have used smtplib module of Python. smtplib is a built-in Python module that provides a way to send email messages using the Simple Mail Transfer Protocol (SMTP). It allows developers to connect to an SMTP server, authenticate themselves, and send email messages to one or more recipients. To send email using smtplib, we first need to setup the sender's mail and password. I have stored all the sensitive information like passwords, API tokens etc. in an .env file. Also, we need to prepare the message to be sent over email subject and email body. The receiver's email address is obtained from the input form. Then we setup smtp\_server and this smtp\_server is used to send the email. The code for sending notifications using email is shown below

```

# function to send email
def send_mail(symbol, threshold, currentPrice, subscriber_email, frequency,
email_type="price_update"):
    try:
        # sender's email
        sender = SENDER_MAIL
        # receiver's email
        recipients = [subscriber_email]
        # Email Content
        if email_type == "price_update":
            subject = f"Stock Price Alert for {symbol}"
            body = f"""
            <h2> Notification for {symbol} </h2>
            <p>Current Price of {symbol} is {currentPrice}. It has crossed
the threshold value of {threshold} set by you.</p>
            <br>
            This is the Way 📊💰
            """
            elif email_type == "new_entry":
                subject = f"Subscription Added for {symbol}"
                body = f"""
                <p>Dear Customer, you have successfully added {frequency}
subscription for {symbol}. Your current price threshold is {threshold}. You
will get notified if the current price exceeds this threshold.</p>
                <br>
                This is the Way 📊💰
                """
            # using MIME on body to send html content
            msg = MIMEText(body, 'html')
            msg['Subject'] = subject
            msg['From'] = sender
            # for sending to multiple recipients
            msg['To'] = ', '.join(recipients)
            # Setup SMTP server
            smtp_server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
            smtp_server.login(SENDER_MAIL, SENDER_PWD)
            smtp_server.sendmail(sender, recipients, msg.as_string())
            smtp_server.quit()
            print(f"Email sent successfully to {subscriber_email}")
        except Exception as e:
            print(f"Error sending email to {subscriber_email}")
            print(f"Following Exception Occured: ")
            print(e)

```

## B. Sending Text Message

To send text message to subscribers, I have used Twilio's free trial account. Twilio provides a free trial account with an API and number that can be used to send text message to numbers. The authentication token, Twilio sid and Twilio number were stored in .env file and this Twilio number will be used as sender's number. The receiver's number is obtained from the input form. To send the text message, we first need to create the text message and then send the message using Twilio API. The code for sending text message is shown below:

```
# function to send message
def send_message(symbol, threshold, currentPrice,
subscriber_number,frequency,msg_type="price_update"):
    try:
        # set up twilio api
        client = Client(TWILIO_SID,TWILIO_AUTH_TOKEN)
        # message when price is updated and exceeds threshold
        if msg_type == "price_update":
            message = client.messages.create(
                from_=TWILIO_NUMBER,
                body=f"Current Price of {symbol} is {currentPrice}. It has
crossed the threshold value of {threshold} set by you.\n~ This is the Way",
                to=subscriber_number
            )
        # Default message when user adds new entry
        elif msg_type == "new_entry":
            message = client.messages.create(
                from_=TWILIO_NUMBER,
                body=f"Dear Customer, you have successfully added
{frequency} subscription for {symbol}. Your current price threshold is
{threshold}. You will get notified if the current price exceeds this
threshold.\n~ This is the Way",
                to=subscriber_number
            )
        print(f"Message sent successfully to {subscriber_number}")
    except Exception as e:
        print(f"Error sending email to {subscriber_number}")
        print(f"Following Exception Occured: ")
        print(e)
```

## 7. Redirect to subscription success page:

After, user adds a new subscription in web app, the user is redirected to a new page which displays the message that subscription was added and also displays the list of all available subscribers. Also, an email / text message will be sent to the user notifying them that new subscription was added successfully. The subscription success page is shown below. Also other outputs are shown in the outputs section.

### SUBSCRIPTION ADDED

You have successfully subscribed to the stock price notifier for the stock "META". You will receive an email when the price of the META stock rises above 220.0.

[Back To Home](#)

### ACTIVE SUBSCRIBERS

Email	Phone Number	Symbol	Threshold	Frequency	Notification Mode
sabo@temp-inbox.me	+11234567890	META	220.0	minute	email
sabo@temp-inbox.me	+11234567890	GOOGL	120.0	day	email
ace@temp-inbox.me	+9779861208287	TSLA	150.0	minute	email

Figure 3: Subscription success page after user clicks on submit

## 2. OUTPUT SCREENSHOTS

### A. Email notification for new subscription



#### Subscription Added for TSLA

From:  
asrdevnepali@gmail.com

Received At: April 22, 2023,  
01:48

Dear Customer, you have successfully added minute subscription for TSLA. Your current price threshold is 150.0. You will get notified if the current price exceeds this threshold.

This is the Way 🙌👉

### B. Email notification for stock price alert



#### Stock Price Alert for TSLA

From:  
asrdevnepali@gmail.com

Received At: April 22, 2023, 01:48

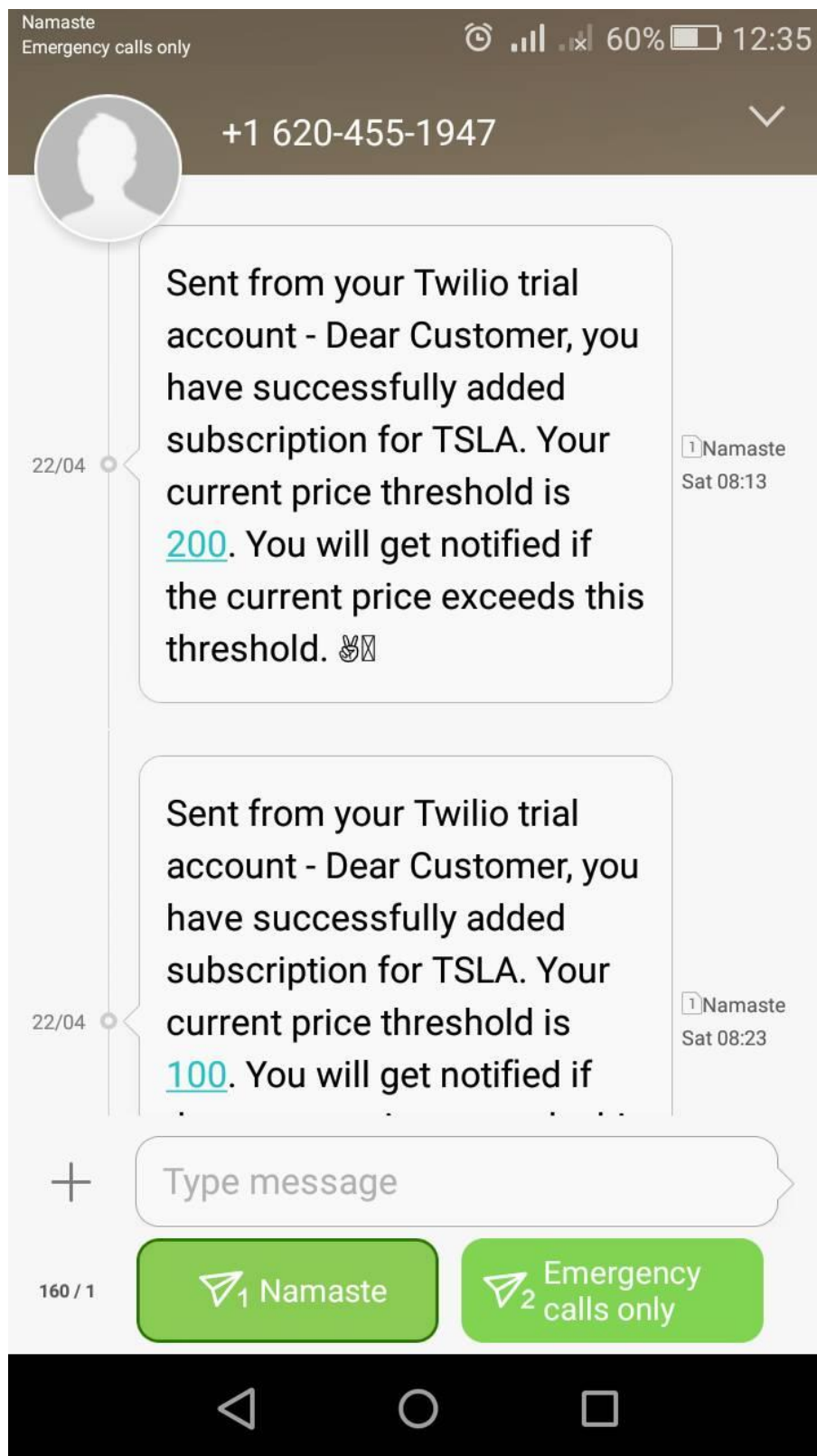
#### Notification for TSLA

Current Price of TSLA is 165.08. It has crossed the threshold value of 150.0 set by you.

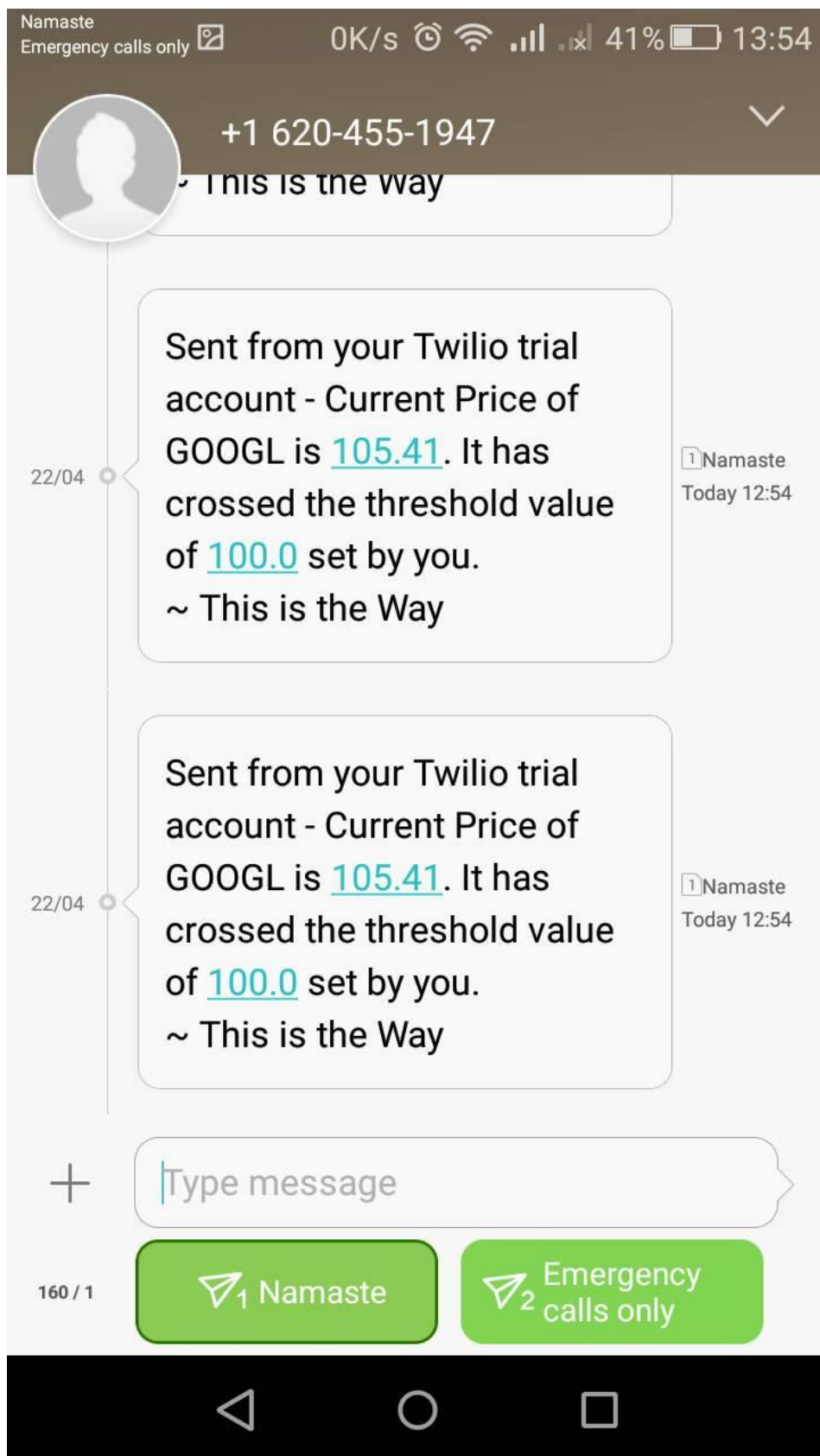
This is the Way 🙌👉



### C. Text message notification for new subscription



## D. Text message notification for stock price alert



## CONCLUSION

To conclude, the challenge project for “Yahoo Stock Price Notifier” was developed using Python and Flask framework and is working successfully. In this project, a flask web application was developed, where users can subscribe for a stock price by choosing the stock price symbol, stock price threshold, frequency and notification mode. Based on the frequency and notification mode, the web app will notify the subscriber when the price of stock exceeds the stock price threshold set by user. Also, a simple sqlite3 database was used in this project so store the details of subscriber. The outputs for this project can be seen in outputs section. All necessary code for this project has been uploaded in GitHub. All the necessary objectives and tasks mentioned in the task description has been successfully implemented and I am planning to add more features in this web app.

The following are some future enhancements possible in this project.

1. Adding user authentication system to get user's details like email and number during registration.
2. Adding feature to unsubscribe.
3. Better UI and exception handling.

GitHub Project Link: [JuJu2181/Stock-Price-Notifier: A simple data engineering project for notifying user about stock prices \(github.com\)](https://github.com/JuJu2181/Stock-Price-Notifier)