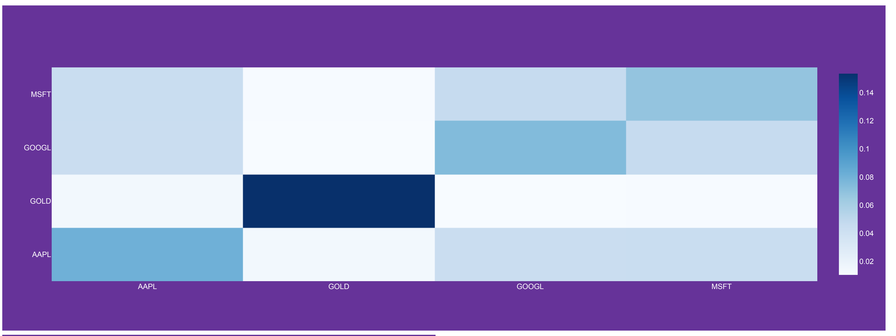
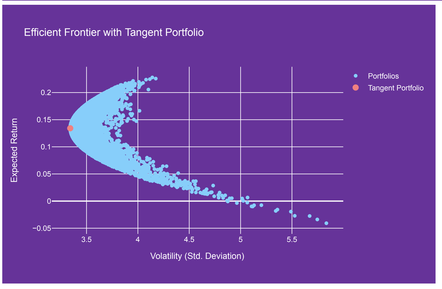
In [1]:
```
import yfinance as yf
import numpy as np
import pandas as pd
from scipy.optimize import minimize
import plotly.graph_objects as go

def portfolio_optimization(tickers, startdate, enddate, frequency, risk_free):
    data = yf.download(tickers, start=startdate, end=enddate, interval=frequency)['Adj Close']
    log_returns = np.log(data / data.shift(1))

    risk_free_rate = risk_free if not isinstance(risk_free, str) else np.mean(np.log(yf.download(risk_free, start=startdate, end=enddate, interval=frequency)['Adj Close']).pct_change())

    cov_matrix = log_returns.cov() * 252
    fig_heatmap = go.Figure(data=go.Heatmap(z=cov_matrix, x=cov_matrix.columns, y=cov_matrix.columns, colorscale='Blues'))
    fig_heatmap.update_layout(
        font=dict(family="Arial, sans-serif", size=12, color="White"),
        paper_bgcolor='RebeccaPurple',
        plot_bgcolor='RebeccaPurple'
    )
    fig_heatmap.show()

    def objective(weights):
        return np.dot(weights.T, np.dot(cov_matrix, weights))

    constraints = ({'type': 'eq', 'fun': lambda weights: np.sum(weights) - 1})
    bounds = tuple((0, 1) for _ in range(len(tickers)))
    initial_weights = [1./len(tickers) for _ in tickers]
    optimal_portfolio = minimize(objective, initial_weights, method='SLSQP', bounds=bounds, constraints=constraints)
    optimal_weights = optimal_portfolio.x

    num_portfolios = 10000
    results = np.zeros((3, num_portfolios))
    for i in range(num_portfolios):
        weights = np.random.random(len(tickers))
        weights /= np.sum(weights)
        portfolio_return = np.sum(log_returns.mean() * weights) * 252
        portfolio_stddev = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights))) * np.sqrt(252)
        results[0,i] = portfolio_return
        results[1,i] = portfolio_stddev
        results[2,i] = (results[0,i] - risk_free_rate) / results[1,i]

    fig_ef = go.Figure()
    fig_ef.add_trace(go.Scatter(x=results[1,:], y=results[0,:], mode='markers', name='Portfolios', marker_color='LightSkyBlue'))
    fig_ef.add_trace(go.Scatter(x=[np.sqrt(objective(optimal_weights)) * np.sqrt(252)], y=[np.sum(log_returns.mean() * optimal_weights) * 252], mode='markers', marker=dict(color='Light
    fig_ef.update_layout(
        title='Efficient Frontier with Tangent Portfolio',
        xaxis_title='Volatility (Std. Deviation)',
        yaxis_title='Expected Return',
        autosize=False,
        font=dict(family="Arial, sans-serif", size=12, color="White"),
        paper_bgcolor='RebeccaPurple',
        plot_bgcolor='RebeccaPurple',
        legend=dict(
            x=1,
            y=1,
            bgcolor='rgba(255, 255, 255, 0)',
            bordercolor='rgba(255, 255, 255, 0)'
        )
    )
    fig_ef.show()

    return optimal_weights
```

In [2]:
```
# Set the parameters of the function

tickers = ["GOOGL","AAPL","MSFT","GOLD"]
startdate = '2010-01-01'
enddate = '2023-09-30'
frequency = '1d'
risk_free = 'SPY'
```

In [3]:
```
# Call the function adn print results

optimal_weights = portfolio_optimization(tickers, startdate, enddate, frequency, risk_free)
print("Optimal Weights: ", optimal_weights)
```

```
[**********************100%**********************]  4 of 4 completed
[**********************100%**********************]  1 of 1 completed
```



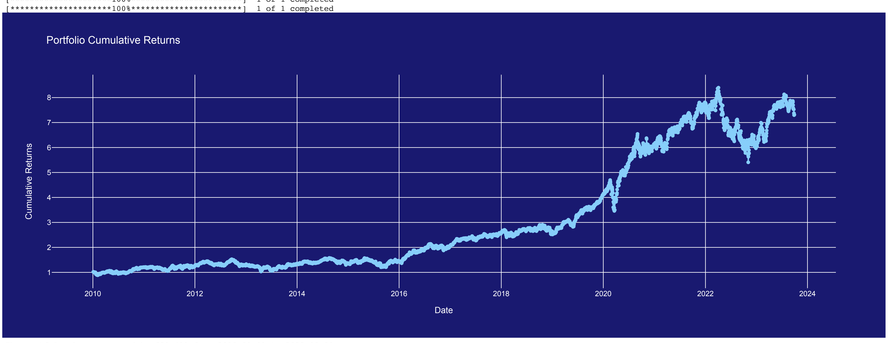



```
Optimal Weights:  [0.19875984 0.23062662 0.26101907 0.30959447]
```

In [4]:
```
# Simulation on the past of hwo the tangent portfolio would have performed


import plotly.graph_objects as go
import yfinance as yf
import pandas as pd
import numpy as np

def plot_portfolio_returns(tickers, start_date, end_date, weights):
    # Fetch historical daily prices and organize them in a DataFrame
    prices = pd.DataFrame()
    for ticker in tickers:
        prices[ticker] = yf.download(ticker, start=start_date, end=end_date)['Adj Close']

    # Compute the daily returns
    returns = prices.pct_change().dropna()

    # Compute the daily portfolio returns
    portfolio_returns = returns.dot(weights)

    # Compute the cumulative portfolio returns
    cumulative_returns = (1 + portfolio_returns).cumprod()

    # Compute the percentage increase of the portfolio
    percentage_increase = (cumulative_returns.iloc[-1] - 1) * 100

    # Initialize the figure
    fig = go.Figure()
    # Customize the color of the line and the markers
    line_color = 'LightSkyBlue'  # Choose your preferred line color
    background_color = 'MidnightBlue'  # Choose your preferred background color
    font_color = 'White'  # Choose your preferred font color

    # Add the portfolio cumulative returns trace to the figure
    fig.add_trace(go.Scatter(
        x=cumulative_returns.index,
        y=cumulative_returns,
        mode='lines+markers',
        name='Portfolio Cumulative Returns',
        line={'color': line_color, 'width': 2}
    ))

    # Update the layout of the figure for better readability and aesthetic
    fig.update_layout(
        title='Portfolio Cumulative Returns',
        xaxis_title='Date',
        yaxis_title='Cumulative Returns',
        font={'family': 'Arial, sans-serif', 'size': 12, 'color': font_color},
        template='plotly',
        plot_bgcolor=background_color,  # Updating the background color of the plot
        paper_bgcolor=background_color  # Updating the background color of the overall figure
    )

    # Display the figure
    fig.show()


    return percentage_increase
```

In [5]:
```
# Call the function and print the portfolio's percentage increase
increase = plot_portfolio_returns(tickers, start_date=startdate, end_date=enddate, weights=optimal_weights)
print(f'The portfolio has increased by {increase:.2f}% from {startdate} to {enddate}.')
```

```
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
[**********************100%**********************]  1 of 1 completed
```




```
The portfolio has increased by 631.41% from 2010-01-01 to 2023-09-30.
```

In [ ]: