Building MPTLab: A React + PHP Modern Portfolio Optimizer with Live Market Data

This walkthrough shows how to build a slick Modern Portfolio Theory (MPT) playground in React that:

- Works with **three types of data sources**:
  - Built-in sample data
  - Uploaded CSVs
  - Live market data fetched from Yahoo Finance via a **PHP proxy**
- Runs a **Monte Carlo simulation** over thousands of random portfolios
- Visualizes the **efficient frontier**, with:
  - Max Sharpe portfolio
  - Global minimum variance portfolio
- Shows optimal **allocations** in charts and a comparison table
- Gracefully falls back to **simulated data** if the live API fails

We'll cover both the **React frontend** and the **PHP API proxy**, plus the MPT math that ties it together.

1. What the App Does

At a high level, the app:

1. Builds a universe of assets (e.g. `AAPL, MSFT, GOOG, TSLA`) from:
   - Sample data
   - CSV upload
   - Live OHLC data via Yahoo Finance
2. Converts price history into daily returns
3. Computes annualized:
   - Expected returns vector
   - Covariance matrix
4. Generates `N` random portfolios (Monte Carlo) and computes for each:
   - Expected portfolio return
   - Portfolio volatility
   - Sharpe ratio
5. Picks:
   - The **max Sharpe ratio** portfolio
   - The **global minimum variance** portfolio
6. Visualizes all simulated portfolios on the **return vs. volatility** plane and highlights the

optimal ones.

2. Tech Stack

- **Frontend**: React + Recharts + Tailwind-style utility classes
- **Icons**: `lucide-react`
- **Backend**: PHP + cURL as a simple proxy to Yahoo Finance's "chart" endpoint

3. Modern Portfolio Theory Math in JavaScript

3.1. Daily Returns

We start from price history of the form:

```
[
  { Date: '2023-01-01', AAPL: 150, MSFT: 230, ... },
  { Date: '2023-01-02', AAPL: 151, MSFT: 232, ... },
  ...
]
```

Daily return for asset i at day t:

$$r_{t,i} = \frac{P_{t,i} - P_{t-1,i}}{P_{t-1,i}}$$

Code:

```
// Calculate simple daily returns: (Price_t - Price_t-1) / Price_t-1
const calculateReturns = (data, tickers) => {
  const returns = [];
  for (let i = 1; i < data.length; i++) {
    const row = {};
    let validRow = true;
    tickers.forEach(ticker => {
      const prev = parseFloat(data[i - 1][ticker]);
      const curr = parseFloat(data[i][ticker]);
      if (isNaN(prev) || isNaN(curr) || prev === 0) {
        validRow = false;
      } else {
        row[ticker] = (curr - prev) / prev;
      }
    });
    if (validRow) returns.push(row);
  }
  return returns;
};
```

We drop rows where any asset's data is invalid so that all return vectors remain aligned.

3.2. Annualized Means and Covariance Matrix

We assume 252 trading days per year.

- Mean daily return: average of returns per asset
- Annualized mean: multiply by 252
- Covariance: standard sample covariance of daily returns, annualized by multiplying by 252

```
const calculateStats = (returns, tickers) => {
  const n = returns.length;
  if (n === 0) return { means: {}, covMatrix: [] };

  // 1. Mean Returns (Annualized)
  const means = {};
  tickers.forEach(t => {
    const sum = returns.reduce((acc, r) => acc + r[t], 0);
    means[t] = (sum / n) * 252;
  });

  // 2. Covariance Matrix (Annualized)
  const covMatrix = Array(tickers.length)
    .fill(0)
    .map(() => Array(tickers.length).fill(0));

  for (let i = 0; i < tickers.length; i++) {
    for (let j = 0; j < tickers.length; j++) {
      let sumProduct = 0;
      const t1 = tickers[i];
      const t2 = tickers[j];
      const mean1 = means[t1] / 252; // daily mean
      const mean2 = means[t2] / 252;

      for (let k = 0; k < n; k++) {
        sumProduct += (returns[k][t1] - mean1) * (returns[k][t2] - mean2);
      }
      covMatrix[i][j] = (sumProduct / (n - 1)) * 252;
    }
  }

  return { means, covMatrix };
};
```

The result:

- `means`: `{ AAPL: 0.12, MSFT: 0.10, ... }` (annualized)
- `covMatrix`: `N x N` matrix of annualized covariances

3.3. Monte Carlo Portfolios and Sharpe Ratio

For each Monte Carlo trial:

1. Draw random weights and normalize them to sum to 1
2. Portfolio return:

$$\mu_p = \sum_i w_i \mu_i$$

3. Portfolio variance:

$$\sigma_p^2 = \mathbf{w}^T \Sigma \mathbf{w}$$

4. Sharpe ratio:

$$S = \frac{\mu_p - r_f}{\sigma_p}$$

Code:

```
const runSimulation = (stats, tickers, riskFreeRate, iterations = 2000) => {
  const results = [];
  const { means, covMatrix } = stats;

  for (let i = 0; i < iterations; i++) {
    // 1. Random weights
    let weights = tickers.map(() => Math.random());
    const sumWeights = weights.reduce((a, b) => a + b, 0);
    weights = weights.map(w => w / sumWeights); // sum to 1

    // 2. Portfolio return (annualized)
    let portReturn = 0;
    weights.forEach((w, idx) => {
      portReturn += w * means[tickers[idx]];
    });

    // 3. Portfolio variance
    let portVar = 0;
    for (let r = 0; r < tickers.length; r++) {
      for (let c = 0; c < tickers.length; c++) {
        portVar += weights[r] * weights[c] * covMatrix[r][c];
      }
    }
    const portVol = Math.sqrt(portVar);

    // 4. Sharpe
    const sharpe = (portReturn - riskFreeRate) / portVol;
```

```
    results.push({
      id: i,
      return: portReturn,
      volatility: portVol,
      sharpe,
      weights
    });
  }

  return results;
};
```

> Note: `riskFreeRate` is treated as a decimal (e.g. `0.02` = 2%). If you prefer the user to type `2`, just divide by 100 on input.

4. Mock History Generator (Fallback When API Fails)

When the Yahoo/Proxy pipeline fails, v1.1 auto-generates a synthetic history using a simple "Geometric Brownian Motion-ish" process.

```
// Used when the PHP API is not available
const generateMockHistory = (tickers) => {
  const n = 252; // 1 year of daily data
  const data = [];
  const now = new Date();

  const prices = {};
  tickers.forEach(t => (prices[t] = 100 + Math.random() * 400));

  for (let i = 0; i < n; i++) {
    const date = new Date(now);
    date.setDate(date.getDate() - (n - i));
    const dateStr = date.toISOString().split('T')[0];

    const row = { Date: dateStr };
    tickers.forEach(t => {
      const volatility = 0.02;
      const drift = 0.0005;
      const change = drift + (Math.random() - 0.5) * 2 * volatility;
      prices[t] = prices[t] * (1 + change);
      row[t] = prices[t];
    });
    data.push(row);
  }
  return data;
};
```

This way, "Fetch mode" never leaves the UI empty even if the backend or Yahoo is down.

5. React Component: State, Data Modes, and Optimization

The main app lives in `PortfolioOptimizer`.

5.1. Core State

```
const [csvData, setCsvData] = useState([]);
const [tickers, setTickers] = useState([]);
const [riskFreeRate, setRiskFreeRate] = useState(0.02);
const [iterations, setIterations] = useState(2500);
const [simResults, setSimResults] = useState([]);
const [isSimulating, setIsSimulating] = useState(false);
const [isFetching, setIsFetching] = useState(false);
const [error, setError] = useState(null);

// Ticker input + mode
const [tickerInput, setTickerInput] = useState("AAPL, MSFT, GOOG, TSLA");
const [dataSourceMode, setDataSourceMode] = useState("sample"); // 'sample'
| 'upload' | 'fetch'

const fileInputRef = useRef(null);
```

- `csvData` is the unified “price history” regardless of source.
- `tickers` is the universe of assets currently loaded.
- `dataSourceMode` determines which UI to show (sample / fetch / upload).

5.2. Data Source Selector (Sample, Fetch, Upload)

The selector is a simple 3-button segmented control:

```
<div className="flex p-1 bg-slate-900 rounded-lg mb-4">
  <button
    onClick={() => setDataSourceMode('sample')}
    className={/* active vs inactive styles */}
  >
    Sample
  </button>
  <button
    onClick={() => setDataSourceMode('fetch')}
    className={/* active vs inactive styles */}
  >
    Fetch
  </button>
  <button
    onClick={() => setDataSourceMode('upload')}
    className={/* active vs inactive styles */}
  >
    Upload
```

```
  </button>
</div>
```

Sample Mode

Loads built-in demo data (SPY, TLT, GLD, BTC):

```
const loadSampleData = () => {
  const mockData = [
    { Date: '2023-01-01', SPY: 380, TLT: 100, GLD: 170, BTC: 16500 },
    // ... more days ...
    { Date: '2023-01-20', SPY: 412, TLT: 104, GLD: 182, BTC: 23500 },
  ];
  setCsvData(mockData);
  setTickers(['SPY', 'TLT', 'GLD', 'BTC']);
  setSimResults([]);
  setError(null);
};
```

Rendered when `dataSourceMode === 'sample'`:

```
{dataSourceMode === 'sample' && (
  <button
    onClick={loadSampleData}
    className="w-full py-2 px-4 bg-slate-700 hover:bg-slate-600 ..."
  >
    <FileText className="w-4 h-4" /> Load Sample Data
  </button>
)}
```

Upload Mode

Allows users to upload a CSV with format:

```
Date,Ticker1,Ticker2,...
2023-01-01,100,200,...
2023-01-02,101,202,...
...
```

Parser:

```
const handleFileUpload = (e) => {
  const file = e.target.files[0];
  if (!file) return;

  const reader = new FileReader();
```

```
  reader.onload = (event) => {
    try {
      const text = event.target.result;
      const lines = text.split('\n').map(l => l.trim()).filter(l => l);
      const headers = lines[0].split(',').map(h => h.trim());

      const assetTickers = headers.slice(1);
      const parsedData = [];

      for (let i = 1; i < lines.length; i++) {
        const values = lines[i].split(',');
        if (values.length === headers.length) {
          const row = { Date: values[0] };
          assetTickers.forEach((t, idx) => {
            row[t] = parseFloat(values[idx + 1]);
          });
          parsedData.push(row);
        }
      }

      setTickers(assetTickers);
      setCsvData(parsedData);
      setSimResults([]);
      setError(null);
    } catch {
      setError("Failed to parse CSV. Ensure format: Date, Ticker1,
Ticker2...");
    }
  };
  reader.readAsText(file);
};
```

Rendered when `dataSourceMode === 'upload'`.

Fetch Mode (Live Data via PHP Proxy + Fallback)

Fetch mode lets the user type symbols (e.g. `AAPL, MSFT, GOOG`) and fetches prices via your PHP proxy:

```
const handleFetchTickers = async () => {
  const rawTickers = tickerInput
    .split(',')
    .map(t => t.trim().toUpperCase())
    .filter(t => t.length > 0);

  if (rawTickers.length < 2) {
    setError("Please enter at least 2 tickers to optimize a portfolio.");
    return;
  }
```

```
  setIsFetching(true);
  setError(null);

  try {
    const fetchedDataByTicker = {};
    const allDates = new Set();

    for (const t of rawTickers) {
      const url = `https://www.aliazary.com/apps/mptlab/api.php?
mode=json_data&symbol=${encodeURIComponent(t)}&range=1y`;
      const response = await fetch(url);

      if (!response.ok) {
        throw new Error(`HTTP ${response.status} while fetching ${t}`);
      }

      const json = await response.json();

      // Expecting Yahoo-style chart JSON
      const chart = json.chart;
      if (!chart || chart.error || !chart.result || !chart.result[0]) {
        throw new Error(`Unexpected data format for ${t}`);
      }

      const result = chart.result[0];
      const timestamps = result.timestamp || [];
      const quotes =
        result.indicators &&
        result.indicators.quote &&
        result.indicators.quote[0] &&
        result.indicators.quote[0].close
          ? result.indicators.quote[0].close
          : [];

      fetchedDataByTicker[t] = {};

      timestamps.forEach((ts, i) => {
        const price = quotes[i];
        if (price != null && !Number.isNaN(price)) {
          const dateStr = new Date(ts * 1000).toISOString().split("T")[0];
          fetchedDataByTicker[t][dateStr] = price;
          allDates.add(dateStr);
        }
      });
    }

    // Merge into rows: { Date, T1, T2, ... }
    const mergedRows = Array.from(allDates)
      .sort()
```

```
      .map(date => {
        const row = { Date: date };
        rawTickers.forEach(t => {
          const value = fetchedDataByTicker[t][date];
          row[t] = typeof value === "number" ? value : null;
        });
        return row;
      });

    if (mergedRows.length === 0) {
      throw new Error("No price data returned from API.");
    }

    setCsvData(mergedRows);
    setTickers(rawTickers);
    setSimResults([]);
  } catch (err) {
    console.error(err);
    setError("Error fetching real data. Falling back to simulated data.");

    const mockHistory = generateMockHistory(rawTickers);
    setCsvData(mockHistory);
    setTickers(rawTickers);
    setSimResults([]);
  } finally {
    setIsFetching(false);
  }
};
```

If anything fails (HTTP error, unexpected JSON, etc.), we log the error and:

- Show a friendly message
- Switch to `generateMockHistory(rawTickers)` so the optimizer still works

6. Running the Optimization and Picking Optimal Portfolios

Trigger:

```
const handleOptimize = () => {
  setIsSimulating(true);
  setTimeout(() => {
    try {
      const returns = calculateReturns(csvData, tickers);
      if (returns.length < 2) throw new Error("Not enough data points to
calculate returns.");

      const stats = calculateStats(returns, tickers);
      const results = runSimulation(stats, tickers, riskFreeRate,
iterations);
```

```
        setSimResults(results);
        setIsSimulating(false);
      } catch (err) {
        setError(err.message);
        setIsSimulating(false);
      }
    }, 100);
  };
```

We use a tiny `setTimeout` so the button can visually flip to “loading” before the heavy computation.

Then we derive:

- Portfolio with **max Sharpe ratio**
- Portfolio with **minimum volatility**

using `useMemo`:

```
  const optimalPortfolios = useMemo(() => {
    if (simResults.length === 0) return null;

    const maxSharpe = simResults.reduce((prev, current) =>
      prev.sharpe > current.sharpe ? prev : current
    );

    const minVol = simResults.reduce((prev, current) =>
      prev.volatility < current.volatility ? prev : current
    );

    return { maxSharpe, minVol };
  }, [simResults]);
```

7. Visualizing the Efficient Frontier with Recharts

7.1. Efficient Frontier Scatter Plot

We map:

- X-axis: portfolio volatility
- Y-axis: expected return
- Each simulation is a point
- Max Sharpe & min variance portfolios are highlighted with distinct shapes

```
  {simResults.length > 0 ? (
    <ResponsiveContainer width="100%" height={450}>
      <ScatterChart margin={{ top: 60, right: 20, bottom: 20, left: 0 }}>
```

```
      <CartesianGrid strokeDasharray="3 3" stroke="#334155" />
      <XAxis
        type="number"
        dataKey="volatility"
        tickFormatter={val => `${(val * 100).toFixed(1)}%`}
        stroke="#94a3b8"
        fontSize={12}
      />
      <YAxis
        type="number"
        dataKey="return"
        tickFormatter={val => `${(val * 100).toFixed(1)}%`}
        stroke="#94a3b8"
        fontSize={12}
      />
      <Tooltip
        cursor={{ strokeDasharray: '3 3' }}
        content={({ active, payload }) => {
          if (active && payload && payload.length) {
            const data = payload[0].payload;
            return (
              <div className="bg-slate-900 border border-slate-600 p-3
rounded shadow-xl text-xs">
                <p className="font-bold text-white mb-2">Portfolio Stats</p>
                <p className="text-slate-300">
                  Return: <span className="text-green-400 font-mono">
                    {(data.return * 100).toFixed(2)}%
                  </span>
                </p>
                <p className="text-slate-300">
                  Volatility: <span className="text-red-400 font-mono">
                    {(data.volatility * 100).toFixed(2)}%
                  </span>
                </p>
                <p className="text-slate-300">
                  Sharpe: <span className="text-blue-400 font-mono">
                    {data.sharpe.toFixed(2)}
                  </span>
                </p>
              </div>
            );
          }
          return null;
        }}
      />

      {/* All simulated portfolios */}
      <Scatter
        name="Portfolios"
        data={simResults}
```

```
          fill="#3b82f6"
          fillOpacity={0.6}
          shape="circle"
        >
          {simResults.map((entry, index) => (
            <Cell
              key={`cell-${index}`}
              fill={entry.sharpe > 1 ? '#60a5fa' : '#1e40af'}
            />
          ))}
        </Scatter>

        {/* Optimal portfolios */}
        {optimalPortfolios && (
          <>
            <Scatter
              name="Max Sharpe"
              data={[optimalPortfolios.maxSharpe]}
              fill="#fbbf24"
              shape="star"
              r={10}
            />
            <Scatter
              name="Min Volatility"
              data={[optimalPortfolios.minVol]}
              fill="#f87171"
              shape="diamond"
              r={10}
            />
          </>
        )}
      </ScatterChart>
    </ResponsiveContainer>
  ) : (
    <div className="h-[450px] flex flex-col items-center justify-center text-
slate-500">
      <TrendingUp className="w-16 h-16 opacity-20 mb-4" />
      <p>Load data and run optimization to view the efficient frontier.</p>
    </div>
  )}
```

7.2. Allocation Bar Charts for Optimal Portfolios

For both **Max Sharpe** and **Global Min Variance**, you show:

- Return, volatility, Sharpe
- Allocation bar chart per asset

Example for max Sharpe:

```
<ResponsiveContainer width="100%" height="100%">
  <BarChart
    data={tickers.map((t, i) => ({
      name: t,
      value: optimalPortfolios.maxSharpe.weights[i]
    }))}
    layout="vertical"
    margin={{ left: 10, right: 30 }}
  >
    <XAxis type="number" hide domain={[0, 1]} />
    <YAxis
      type="category"
      dataKey="name"
      width={40}
      tick={{ fill: '#94a3b8', fontSize: 10 }}
    />
    <Tooltip
      cursor={{ fill: 'transparent' }}
      content={({ active, payload }) => {
        if (active && payload && payload.length) {
          return (
            <div className="bg-slate-900 text-xs px-2 py-1 rounded border
border-slate-700">
              {(payload[0].value * 100).toFixed(1)}%
            </div>
          );
        }
        return null;
      }}
    />
    <Bar dataKey="value" radius={[0, 4, 4, 0]}>
      {tickers.map((entry, index) => (
        <Cell key={`cell-${index}`} fill={COLORS[index % COLORS.length]} />
      ))}
    </Bar>
  </BarChart>
</ResponsiveContainer>
```

7.3. Allocation Comparison Table

Finally, a simple table shows each asset's weight in:

- Max Sharpe portfolio
- Min variance portfolio

```
<table className="w-full text-sm text-left text-slate-400">
  <thead className="text-xs text-slate-300 uppercase bg-slate-900/50">
    <tr>
```

```
      <th className="px-6 py-3">Asset</th>
      <th className="px-6 py-3">Max Sharpe Weight</th>
      <th className="px-6 py-3">Min Vol Weight</th>
    </tr>
  </thead>
  <tbody>
    {tickers.map((t, i) => {
      const maxW = optimalPortfolios.maxSharpe.weights[i];
      const minW = optimalPortfolios.minVol.weights[i];
      return (
        <tr key={t} className="border-b border-slate-700 hover:bg-slate-
700/50">
          <td className="px-6 py-4 font-medium text-white flex items-center
gap-2">
            <span
              className="w-3 h-3 rounded-full"
              style={{ backgroundColor: COLORS[i % COLORS.length] }}
            ></span>
            {t}
          </td>
          <td className={`px-6 py-4 font-mono ${maxW > 0.2 ? 'text-green-
400' : ''}`}>
            {(maxW * 100).toFixed(2)}%
          </td>
          <td className={`px-6 py-4 font-mono ${minW > 0.2 ? 'text-green-
400' : ''}`}>
            {(minW * 100).toFixed(2)}%
          </td>
        </tr>
      );
    })}
  </tbody>
</table>
```

8. PHP Backend: Proxying Yahoo Finance

To fetch real market data from the browser, you use a tiny PHP proxy. This:

- Avoids CORS headaches
- Hides Yahoo's endpoint from the frontend if you want
- Keeps things simple: the React app just calls your own `api.php`

Here's the full PHP code:

```
<?php
// Simple PHP proxy to Yahoo Finance "chart" endpoint
// URL example: /api.php?mode=json_data&symbol=AAPL&range=1y
```

```php
header('Content-Type: application/json');
header('Access-Control-Allow-Origin: *');

$mode   = isset($_GET['mode'])   ? $_GET['mode']   : '';
$symbol = isset($_GET['symbol']) ? $_GET['symbol'] : '';
$range  = isset($_GET['range'])  ? $_GET['range']  : '1y';

if ($mode !== 'json_data' || empty($symbol)) {
    echo json_encode(['error' => 'Invalid parameters']);
    exit;
}

$interval = '1d';
$baseUrl = 'https://query1.finance.yahoo.com/v8/finance/chart/';
$url = $baseUrl . urlencode($symbol) .
       '?range=' . urlencode($range) .
       '&interval=' . urlencode($interval);

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
curl_setopt($ch, CURLOPT_USERAGENT, 'Mozilla/5.0');

$result = curl_exec($ch);
$httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
$curlError = curl_error($ch);
curl_close($ch);

if ($result === false || $httpCode !== 200) {
    echo json_encode([
        'error'   => 'Upstream request failed',
        'code'    => $httpCode,
        'details' => $curlError
    ]);
    exit;
}

echo $result;
```

How it works:

1. Validates that `mode=json_data` and `symbol` are present.
2. Builds the Yahoo Finance chart URL, e.g.:
   - `https://query1.finance.yahoo.com/v8/finance/chart/AAPL?range=1y&interval=1d`
3. Uses cURL to fetch data, sets a user agent, follows redirects.
4. If anything fails:

- Returns JSON with `error`, `code`, and `details`.

5. If successful:
   - Echoes Yahoo's JSON directly to the frontend.

The frontend expects the typical Yahoo "chart" format:

```
{
  "chart": {
    "result": [
      {
        "timestamp": [...],
        "indicators": {
          "quote": [
            { "close": [...] }
          ]
        }
      }
    ]
  }
}
```

Your `handleFetchTickers` function parses this shape.