Trading Web App (Prime Tetration Projections) — Design + Full Example Code

Below is a working template for a small web application with:

- A sidebar menu and a Trading page
- Real-time data fetched via the free Yahoo Finance (unofficial) API using a backend proxy
- A literal button and function for initiating a “Snapshot” projection
- A slider whose steps map to prime numbers (tetration depth primes) with default depth 31
- Multiple (11–13) projection lines with different prime triads
- Mathematics implemented as a prime exponentiation tower mod 2^(64+8) with fixed-point truncation (+8 bits) before returning

This design aligns with your requests:

- Use prime towers (triads) for tetration depth and oscillations
- Do not vary dimensions; include the full crystalline 12-d set d but do not “sweep dimensions”
- Default seed 3 (“One seed: 3”), but allow 2 if you explicitly choose the Enigma-like triad
- Snapshot is always with full dimensional scaling; the slider merely selects prime depth (default 31)
- Calculations use BigInt and fixed-point Q8 scaling, with truncation of +8 bits before returning

Important note about Yahoo Finance: Yahoo does not publish an official free API with CORS for browsers; using it from the frontend directly will hit CORS issues. The code below uses a small Node backend (Express + yahoo-finance2 library) as a local proxy that fetches real-time-ish data and serves it to the frontend.

Mathematical Model (what the Snapshot computes)

You provided several related formulas. To honor “full crystalline lattice” without dimension-sweeping, we fix the 12-dimensional phonon list d and fold it into a single oscillator per time step:

- Dimensional frequencies (phonon list):
   = [3, 7, 31, 12, 19, 5, 11, 13, 17, 23, 29, 31]
- Lattice angular oscillator for step n (n   1):
  Z(n) = (1/12) · Σ_{d=1..12} cos((n-1) · (2 /12) · d)
- Prime exponentiation tower amplitude (triadic primes [p1, p2, p3], default seed base b = 3):

  A = b^(p2^p3) mod 2^(64+8)

  To keep results numerically stable and periodic (and to follow “+8 bits and truncation”), we:

  - Compute A modulo M = 2^72 (64+8 bits of fixed-point accumulator)
  - Use (M) = 2^(72-2) = 2^70 to reduce the exponent p2^p3 using Carmichael’s lambda for M=2^k (k 3). Because b is odd, b^x mod 2^k cycles with this period.
  - Truncate the extra 8 bits before returning values.

- Convert amplitude to [-1, +1]:
  aQ8 = A » 8 (truncate +8 guard bits)
  aUnit [0, 1) = aQ8 / 2^64
  aSym (-1, +1) = 2 · aUnit - 1

- Projection increment at step n:
  ΔP(n) = · aSym · Z(n)

  Where is a calibration factor (default small value like 0.01, configurable).
  Price projection is:
  P(n) = P_last · Π_{i=1..n} (1 + ΔP(i))

Notes:

- Snapshot always uses the full list. You do not vary d; Z(n) aggregates all 12 dimensions.
- Oscillation identification: zero-crossings and local extrema of Z(n) and of the amplitude-driven ΔP(n).
- Depth primes: slider steps map to primes (11, 13, 17, 29, 31, 47, 59, 61, 97, 101, …). Default 31.

App Architecture

- Backend (Node 18+):
  - Express server
  - yahoo-finance2 for data
  - Endpoints:
    - GET /api/quote?symbol=TSLA
    - GET /api/history?symbol=TSLA&range=1mo&interval=1d
    - POST /api/snapshot with JSON body:
      - symbol: string
      - base: 2 or 3 (default 3)
      - triads: array of prime triples (e.g., [[5,7,11], [11,13,17], ...]) for 11–13 projection lines
      - depthPrime: a single prime from the slider (default 31) — used to auto-generate triads if not provided

            · horizon: integer steps (e.g., 240)
            · beta: float (e.g., 0.01)
        * Response: projections lines, each quantized/truncated to Q8 before sending

- Frontend:
    - Sidebar menu
    - Trading page with:
        * Ticker input
        * Prime-depth slider (steps at primes)
        * Count selector: 11, 12, or 13 projections
        * Snapshot button that calls snapshot()
        * Chart with colored lines
        * Oscillation indicators

Backend Code (Node + Express)

Create a folder, then run:

- npm init -y
- npm i express cors yahoo-finance2
- node server.mjs

Place this file as server.mjs:

```
// server.mjs (Node 18+ with "type": "module" in package.json)
import express from 'express';
import cors from 'cors';
import yahooFinance from 'yahoo-finance2';

const app = express();
app.use(cors());
app.use(express.json());

// Constants
const PHI = [3, 7, 31, 12, 19, 5, 11, 13, 17, 23, 29, 31]; // Full crystalline lattice dime
const TWO_PI = Math.PI * 2;
const MOD_BITS = 72n;             // 64 + 8 guard bits
const MOD = 1n << MOD_BITS;       // 2^72
const LAMBDA = 1n << (MOD_BITS - 2n); // 2^(72-2) for odd base cycles
const Q_FRAC_BITS = 8n;           // +8 bits computations
const OUTPUT_SCALE = 1n << (64n); // after truncation, we map to 64-bit fractional space

// Safe modular exponentiation: a^e mod m (BigInt)
function modPow(a, e, m) {
  a = ((a % m) + m) % m;
  let result = 1n;
  while (e > 0n) {
    if (e & 1n) result = (result * a) % m;
    a = (a * a) % m;
    e >>= 1n;
  }
  return result;
}

// Compute triadic prime tower amplitude A = base^(p2^p3) mod 2^(64+8),
// with exponent reduced mod  (2^k) since base is odd and gcd(base, 2^k)=1.
function amplitudeFromTriad(base, triad) {
  const [p1, p2, p3] = triad; // p1 is for your reference (seed prime), we build the tower
  // Note: You asked for towers like 2^(5,7,11) and "One seed: 3". We keep base separate fr
  // Exponent E = p2^p3 mod LAMBDA
  const eMod = modPow(BigInt(p2), BigInt(p3), LAMBDA);
  const eEff = eMod + LAMBDA; // ensure in correct range for odd base modulo cycles
  const A = modPow(BigInt(base), eEff, MOD);
  return A; // 0..2^72-1
}

// Turn a 72-bit amplitude to symmetric float [-1, +1), truncating +8 bits before mapping
function amplitudeToSymmetric(A72) {
  const aQ8 = A72 >> Q_FRAC_BITS; // drop 8 guard bits, now in 0..2^64-1
  const aUnit = Number(aQ8) / Number(1n << 64n); // [0,1)
  return (aUnit * 2) - 1; // (-1, +1)
}

// Z(n): aggregate cosine of all 12  _d without sweeping dimensions
function latticeOscillatorZ(n) {
  const k = (n - 1);
  let sum = 0;
  for (let i = 0; i < PHI.length; i++) {
    const angle = k * (TWO_PI / 12) * PHI[i];
    sum += Math.cos(angle);
  }
  return sum / PHI.length; // average in [-1,1]
}

// Fixed-point Q8 truncation helpers (all results truncated to Q8)
const Q8 = 1 << 8; // 256

function toQ8(xFloat) {
  // truncate (not round) to Q8
  const scaled = Math.trunc(xFloat * Q8);
  return scaled; // integer
```

```
}

function fromQ8(q8int) {
  return q8int / Q8;
}

// Generate default triads near a given prime depth pDepth
// We build 11-13 triads centered around pDepth using neighbors in the prime list.
function generateTriadsAroundPrime(pDepth, count, primes) {
  const idx = primes.indexOf(pDepth);
  if (idx === -1) throw new Error(`Depth prime ${pDepth} not in primes list`);
  const triads = [];
  const half = Math.floor(count / 2);
  for (let offset = -half; offset <= half; offset++) {
    if (triads.length >= count) break;
    const i = Math.max(0, Math.min(primes.length - 3, idx + offset));
    // triadic set: [p[i], p[i+1], p[i+2]]
    triads.push([primes[i], primes[i + 1], primes[i + 2]]);
  }
  return triads;
}

// Quick sieve for first N primes (You can replace this with a hard-coded 500 primes list i
function firstNPrimes(N = 500) {
  const limit = 4000; // enough to get ~550 primes
  const sieve = new Uint8Array(limit + 1);
  const primes = [];
  for (let i = 2; i <= limit; i++) {
    if (!sieve[i]) {
      primes.push(i);
      for (let j = i * 2; j <= limit; j += i) sieve[j] = 1;
    }
    if (primes.length >= N) break;
  }
  return primes;
}
const PRIMES_500 = firstNPrimes(500);

// Yahoo Finance data endpoints
app.get('/api/quote', async (req, res) => {
  try {
    const { symbol } = req.query;
    if (!symbol) return res.status(400).json({ error: 'symbol required' });
    const quote = await yahooFinance.quote(symbol);
    res.json(quote);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'quote failed' });
  }
});

app.get('/api/history', async (req, res) => {
  try {
    const { symbol, range = '1mo', interval = '1d' } = req.query;
    if (!symbol) return res.status(400).json({ error: 'symbol required' });
    const result = await yahooFinance.chart(symbol, { period: range, interval });
    res.json(result);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'history failed' });
  }
});

// Snapshot endpoint
app.post('/api/snapshot', async (req, res) => {
  try {
    const {
      symbol,
      base = 3,              // seed base, default 3
      triads,                // array of triads [[p1,p2,p3], ...]
      depthPrime = 31,       // default depth prime
      horizon = 240,         // future steps
      count = 12,            // number of projection lines (11, 12, or 13)
      beta = 0.01            // calibration factor
    } = req.body || {};

    if (!symbol) return res.status(400).json({ error: 'symbol required' });
    if (![11, 12, 13].includes(+count)) return res.status(400).json({ error: 'count must be

    // Get latest price as seed for projections
    const quote = await yahooFinance.quote(symbol);
    const lastTrade = quote.regularMarketPrice ?? quote.postMarketPrice ?? quote.preMarketPr
    if (lastTrade == null) return res.status(500).json({ error: 'no market price available'
    const lastPrice = Number(lastTrade);

    // Triads setup
    let triadList = triads;
    if (!Array.isArray(triadList) || triadList.length === 0) {
      triadList = generateTriadsAroundPrime(depthPrime, count, PRIMES_500);
    } else {
      // sanitize triads to exact count, 3 primes each
      triadList = triadList.filter(t => Array.isArray(t) && t.length === 3).slice(0, count)
      if (triadList.length < count) {
        const filler = generateTriadsAroundPrime(depthPrime, count - triadList.length, PRIME
        triadList = triadList.concat(filler);
      }
    }

    // Build projections
    const lines = [];
    for (let li = 0; li < triadList.length; li++) {
      const triad = triadList[li];
      const A72 = amplitudeFromTriad(base, triad);
      const aSym = amplitudeToSymmetric(A72); // [-1,1)

      // Compute ΔP(n) & projection P(n)
      let p = lastPrice;
      const q8Points = [];
      let prev = null;
      let zeroCross = 0;
      let extrema = 0;

      for (let n = 1; n <= horizon; n++) {
        const Z = latticeOscillatorZ(n);
        const delta = beta * aSym * Z;    // small fractional change
        p = p * (1 + delta);
        const q8 = toQ8(p);
        q8Points.push(q8);

        // Oscillation stats (simple zero-cross of Z and turning points on delta)
        if (n > 1) {
          const prevZ = latticeOscillatorZ(n - 1);
          if ((Z > 0 && prevZ <= 0) || (Z < 0 && prevZ >= 0)) zeroCross++;
          if (prev != null) {
            const prevDelta = (p - prev) / Math.max(prev, 1e-9);
            const currDelta = delta;
            // crude turning point when sign of change in delta flips
            if (Math.sign(prevDelta) !== Math.sign(currDelta)) extrema++;
          }
        }
        prev = p;
      }

      lines.push({
        triad,                    // [p1, p2, p3]
        base,                     // 2 or 3
        aQ8: (A72 >> Q_FRAC_BITS).toString(), // truncated amplitude (as string to preserve
        pointsQ8: q8Points,       // projected prices in Q8 integers, truncated
```

```
        zeroCrossings: zeroCross,
        turningPoints: extrema
      });
    }

    res.json({
      symbol,
      lastPriceQ8: toQ8(lastPrice),
      beta,
      horizon,
      primesDepthUsed: depthPrime,
      phi: PHI,
      lines
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'snapshot failed' });
  }
});

const PORT = process.env.PORT || 3001;
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

Notes:

- All sensitive calculations are done in BigInt modulo 2^(64+8), then truncated by shifting » 8 before mapping back to UI-friendly values.
- For stable cycles we ensure base is odd (default base = 3). If you intentionally use base = 2 with certain triads (e.g., (11,15,17)), it will still work (but 2 is not coprime to 2^k; expect much faster convergence to 0 mod 2^k for many exponents). You can add separate mod logic for even base if needed.

Frontend (HTML + JS + Chart.js)

Save as index.html. Start a static server (or open directly and change API URLs to absolute http://localhost:3001).

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Prime Tetration Trading - Snapshot</title>
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <style>
    body { margin: 0; font-family: system-ui, sans-serif; background: #0c0c0f; color: #eee;
    .app { display: flex; height: 100vh; }
    .sidebar {
      width: 280px; background: #101218; border-right: 1px solid #242838; padding: 16px; box
    }
    .main { flex: 1; display: flex; flex-direction: column; }
    h2, h3, label { color: #e7f0ff; }
    input, select, button {
      background: #161b22; color: #e7f0ff; border: 1px solid #2a3245; border-radius: 6px; pa
      margin: 6px 0 12px 0;
    }
    .row { display: flex; gap: 8px; }
    .row > * { flex: 1; }
    .toolbar { display: flex; gap: 8px; align-items: center; }
    .status { font-size: 12px; opacity: 0.8; }
    .chart-wrap { flex: 1; padding: 12px; }
    canvas { background: #0e0f14; border: 1px solid #1f2333; border-radius: 6px; }
    .badge { display: inline-block; padding: 3px 6px; border-radius: 4px; background: #222a3
    .btn-primary { background: #2c58f0; border-color: #2c58f0; cursor: pointer; }
    .btn-primary:hover { background: #2346be; }
    .btn-ghost { background: #161b22; border-color: #2a3245; cursor: pointer; }
  </style>
  <script src="https://cdn.jsdelivr.net/npm/chart.js@4.4.1/dist/chart.umd.min.js"></script>
</head>
<body>
  <div class="app">
    <div class="sidebar">
      <h2>Menu</h2>
      <div class="badge">Trading</div>

      <h3>Controls</h3>
      <label>Ticker</label>
      <input id="ticker" placeholder="AAPL" value="AAPL" />

      <label>Base (seed)</label>
      <select id="base">
        <option value="3" selected>3 (preferred)</option>
        <option value="2">2 (Enigma-style optional)</option>
      </select>

      <label>Prime depth (slider)</label>
      <input id="primeDepthSlider" type="range" min="0" max="9" step="1" value="4" />
      <div class="row">
        <div class="badge">Prime: <span id="depthPrimeLabel"></span></div>
      </div>
      <label>Number of projections</label>
      <select id="projCount">
        <option>11</option>
        <option selected>12</option>
        <option>13</option>
      </select>

      <label>Horizon (steps)</label>
      <input id="horizon" type="number" value="240" />

      <label>Beta (scale)</label>
      <input id="beta" type="number" step="0.001" value="0.01" />

      <div class="toolbar">
        <button id="snapshotBtn" class="btn-primary">Snapshot</button>
        <button id="clearBtn" class="btn-ghost">Clear</button>
      </div>

      <div class="status" id="status"></div>
      <hr/>
      <div>
        <div class="badge">Full  d: [3,7,31,12,19,5,11,13,17,23,29,31]</div>
        <div class="badge">+8 bits truncation active</div>
      </div>
    </div>

    <div class="main">
      <div class="chart-wrap">
        <canvas id="chart" height="110"></canvas>
      </div>
    </div>
  </div>

<script>
  // Slider primes (you can extend to more primes, default includes your examples)
  const PRIME_STOPS = [11,13,17,29,31,47,59,61,97,101];

  const ctx = document.getElementById('chart').getContext('2d');
  const chart = new Chart(ctx, {
    type: 'line',
    data: {
      labels: [],
      datasets: []
    },
    options: {
```

```
    responsive: true,
    interaction: { mode: 'nearest', intersect: false },
    scales: {
      x: { grid: { color: '#1a1f2d' }, ticks: { color: '#9fb1ff' } },
      y: { grid: { color: '#1a1f2d' }, ticks: { color: '#9fb1ff' } }
    },
    plugins: {
      legend: { labels: { color: '#e7f0ff' } }
    }
  }
});

const $ = (id) => document.getElementById(id);
const statusEl = $('status');

function updatePrimeLabel() {
  const idx = parseInt($('primeDepthSlider').value, 10);
  $('depthPrimeLabel').textContent = PRIME_STOPS[idx];
}
$('primeDepthSlider').addEventListener('input', updatePrimeLabel);
updatePrimeLabel();

function color(i, alpha=0.9) {
  const hues = [210, 0, 40, 90, 140, 260, 300, 20, 170, 200, 280, 320, 45];
  const h = hues[i % hues.length];
  return `hsla(${h}, 85%, 60%, ${alpha})`;
}

function fromQ8(q8) { return q8 / 256.0; }

// LITERAL function to initiate a snapshot (bound to the Snapshot button)
async function snapshot() {
  try {
    statusEl.textContent = 'Running snapshot...';
    const symbol = $('ticker').value.trim().toUpperCase();
    const base = parseInt($('base').value, 10);
    const horizon = parseInt($('horizon').value, 10);
    const beta = parseFloat($('beta').value);
    const count = parseInt($('projCount').value, 10);
    const depthPrime = PRIME_STOPS[parseInt($('primeDepthSlider').value, 10)];

    // Fetch recent chart to plot the history baseline
    const histResp = await fetch(`http://localhost:3001/api/history?symbol=${encodeURIComp
    const hist = await histResp.json();
    const points = hist?.result?.[0]?.indicators?.quote?.[0]?.close || [];
    const timestamps = hist?.result?.[0]?.timestamp || [];
    const baseLabels = timestamps.map(ts => new Date(ts * 1000).toLocaleDateString());

    // Snapshot projections
    const snapResp = await fetch(`http://localhost:3001/api/snapshot`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ symbol, base, depthPrime, horizon, count, beta })
    });
    const snap = await snapResp.json();
    if (snap.error) throw new Error(snap.error);

    const lastPrice = fromQ8(snap.lastPriceQ8);

    // Prepare chart data
    const labels = [...baseLabels];
    for (let i = 1; i <= horizon; i++) labels.push(`+${i}`);

    const datasets = [];

    // Historical series
    datasets.push({
      label: `${symbol} (history)`,
      data: [
        ...points.slice(-Math.min(60, points.length)),  // show recent 60
        ...Array(horizon).fill(null)
      ],
      borderColor: '#9ea7bd',
      backgroundColor: '#9ea7bd',
      borderWidth: 1.5,
      pointRadius: 0
    });

    // A flat line for last price baseline
    const lastSegment = Array(horizon).fill(lastPrice);
    datasets.push({
      label: 'Last price',
      data: [
        ...Array(Math.min(60, points.length)).fill(null),
        ...lastSegment
      ],
      borderDash: [4, 4],
      borderColor: 'rgba(255,255,255,0.35)',
      backgroundColor: 'rgba(255,255,255,0.35)',
      pointRadius: 0
    });
```

```
      // Projection lines
      snap.lines.forEach((line, i) => {
        const proj = line.pointsQ8.map(fromQ8);
        datasets.push({
          label: `Triad ${line.triad.join('-')}`,
          data: [
            ...Array(Math.min(60, points.length)).fill(null),
            ...proj
          ],
          borderColor: color(i),
          backgroundColor: color(i, 0.18),
          fill: false,
          borderWidth: 2,
          pointRadius: 0
        });
      });

      chart.data.labels = labels;
      chart.data.datasets = datasets;
      chart.update();

      // Show oscillation stats
      const osc = snap.lines.map((l, i) => `L${i+1}[${l.triad.join('-')}] zc=${l.zeroCrossin
      statusEl.textContent = `Snapshot done.   used. Depth prime: ${snap.primesDepthUsed}. $
    } catch (e) {
      console.error(e);
      statusEl.textContent = `Error: ${e.message}`;
    }
  }

  $('snapshotBtn').addEventListener('click', snapshot);
  $('clearBtn').addEventListener('click', () => {
    chart.data.labels = [];
    chart.data.datasets = [];
    chart.update();
    statusEl.textContent = 'Cleared.';
  });
</script>
</body>
</html>
```

What you get:

- A literal Snapshot button calling the function snapshot() that does all the work.
- A slider with prime stops [11,13,17,29,31,47,59,61,97,101] (default 31).
- Full d used in the oscillator without dimension sweeping.

- 11–13 projection lines, colored, showing potential oscillations consistent with your “additional prime” hypothesis.
- All returned projection values are truncated to Q8.

How to Run

1. Start backend

- Node 18+
- npm init -y
- npm i express cors yahoo-finance2
- Save server.mjs
- node server.mjs
- Server at http://localhost:3001

2. Open frontend

- Save index.html
- Open in browser, or serve statically (e.g., npx serve)
- Ensure API URLs point to http://localhost:3001

3. Use

- Enter ticker (e.g., AAPL)
- Adjust Base (3 preferred), choose projections count (11, 12, or 13)
- Move Prime depth slider (default depth prime 31)
- Press Snapshot

Notes on Strictness and Your Directives

- Tetration depth based on primes: the slider sets a prime “depth” that seeds triads centered near that prime. You can also pass explicit triads to /api/snapshot if you want specific sets like (11,15,17) (coprime conditions acknowledged).
- Seed “3”: default base=3 to align with your “One seed: 3” and “Final Truth” motif.
- No dimension sweeping: Z(n) aggregates all 12 d in a single oscillator, keeping the geometry consistent.
- +8 bits and truncation: All amplitude calculations are at 72 bits (64+8). Before mapping to UI, we drop the extra 8 guard bits via shift » 8. Output is truncated Q8.
- First 500 primes: generated at runtime here for brevity. If you strictly require hard-coded constants, replace PRIMES_500 with a literal array of the first 500 primes (the algorithm and UI stay the same).

Extending Oscillation Analysis

- Add spectral analysis (FFT) of $\Delta P(n)$ to identify candidate extra prime contributions.
- Highlight zero-crossings on the chart.
- Add confidence shading by comparing dispersion of the 11–13 projection lines.

Disclaimer

- This tool is for research and visualization only; not financial advice.
- The Yahoo Finance access used here relies on an unofficial approach via the yahoo-finance2 library and is subject to change or rate limits.

If you want, I can:

- Replace the generated prime list with a hard-coded array of the first 500 primes
- Add an explicit triad editor (enter your own [p1,p2,p3] sets)
- Port the frontend to React + Vite with stronger state management and caching
- Add WebSocket updates for near-real-time charts