K6 Framework End-to-End Structure

1. Configuration Layer

The configuration layer centralizes all runtime parameters to ensure consistency across environments.

- The file `config/default.json` contains all default runtime parameters:
  - Target `baseUrl`
  - Load profile (`options.vus`, `options.duration`, and thresholds)
  - Output paths for reports
  - Default pacing (`sleepBetweenIterations`)
- The script `scripts/lib/config.js` loads this JSON configuration, freezes it, and provides helper functions:
  - `getBaseUrl()` and `getSleepDuration()` allow overriding defaults through environment variables such as `BASE_URL` and `SLEEP_INTERVAL`.
  - `buildOptions(overrides)` merges configuration options with runtime overrides supplied when launching k6.

2. Test Script

The main test logic is defined in `tests/smoke-test.js`, which imports configuration helpers and defines the smoke scenario.

- `export const options = buildOptions();` integrates the configuration-derived load settings.
- The `setup()` function retrieves dynamic values such as base URL and sleep interval, providing them to each Virtual User (VU).
- The `default` function orchestrates the actual test flow:
  - Each endpoint (`landing`, `contact`, `news`) is tested within a `group`.
  - Each group performs an `http.get` request, records timing metrics in a custom `Trend`, and verifies health via `check()`.
  - After executing all groups, the script pauses for the configured duration to maintain pacing.
- The `handleSummary()` function aggregates and forwards final test results to the `buildSummary()` function for reporting.

3. Execution Flow

Executing `k6 run tests/smoke-test.js` triggers the following sequence:

1. K6 loads the `options` to determine virtual users (VUs), duration, and thresholds.
2. It invokes `setup()` once to build the context object (`baseUrl`, `sleepDuration`).
3. It spawns the configured number of VUs, each repeatedly executing the `default` function for the defined duration.
4. Each VU issues three HTTP requests per iteration, recording performance metrics.
5. Thresholds (e.g., P95 latency ¡ 500ms) are applied to determine test pass or fail.
6. Upon completion, K6 calls `handleSummary(data)` to consolidate and format the results.

4. Reporting Pipeline

The summary reporting mechanism is handled by `scripts/lib/summary.js`.

- It formats summary data according to the paths defined in `config`, such as `summary.json` and `summary.html`.
- It returns an object compatible with K6, e.g.,

```
{ 'stdout': text, 'reports/smoke-summary.html': htmlContent }
```

- The JSON report (`data/results/smoke-summary.json`) stores raw performance metrics such as `http_reqs`, durations, and thresholds.
- The HTML report (`reports/smoke-summary.html`) presents a human-readable visualization of the same data, with charts and summary tables.

5. Custom Metrics and Thresholds

The K6 framework provides flexibility for defining custom metrics.

- Trend metrics (e.g., `landing_page_duration`) record timing per endpoint.
- Endpoint-specific thresholds can be set under `options.thresholds` in the configuration file.
- Built-in metrics like `http_reqs`, `http_req_duration`, and `checks` are automatically captured.
- Default thresholds for common metrics are already predefined in `default.json`.

6. Extending or Tweaking the Framework

The framework is modular and easily extendable.

- New scenarios or endpoints can be added by expanding the `default` function or defining new K6 `scenarios` through:

```
buildOptions({ scenarios: {...} })
```

- Load patterns, such as ramping VUs or arrival-rate executors, can be configured in `config/default.json` or supplied as overrides.
- The reporting logic can be customized by modifying `scripts/lib/summary.js` to support new output formats or destinations.

7. Net Effect

The configuration layer governs load parameters, the test script executes the load test based on these configurations, and the summary module consolidates all metrics into comprehensive JSON and HTML reports. This structure ensures transparency, scalability, and reproducibility for all load testing activities.