# Node.js Web Scraper Project Structure

Web scraping project with Puppeteer/Playwright, organized scrapers, and data pipelines.

#node #nodejs #scraper #web-scraping #puppeteer #playwright

PNG PDF Copy Prompt

## Project Directory

```
myscraper/
    package.json
    tsconfig.json
    README.md
    .gitignore
    .env.example  Proxy settings
  src/
    index.ts  CLI entry
    scrapers/  One per target …
      index.ts
      base.ts  Base scraper cl…
      example-site.ts
    extractors/  Data extraction…
      index.ts
      product.ts
    pipelines/  Data processing
      index.ts
      clean.ts  Data cleaning
      store.ts  CSV/JSON/DB out…
    types/
      index.ts
      item.ts  Scraped data ty…
    utils/
      browser.ts  Browser setup
      retry.ts  Retry logic
      delay.ts  Rate limiting
    config.ts  Settings
  data/  Output directory
    .gitkeep
  tests/
    fixtures/  Saved HTML
    extractors.test.ts
```

## Why This Structure?

Node.js excels at browser automation with Puppeteer or Playwright. This structure separates scrapers (navigation), extractors (data parsing), and pipelines (processing). Each site gets its own scraper module. Testing uses saved HTML fixtures.

## Key Directories

**`scrapers/`** - One module per target site

**`extractors/`** - Extract structured data from pages

**`pipelines/`** - Clean, validate, and store data

**`data/`** - Output CSVs, JSONs, or database files

## Playwright Scraper

```ts
// src/scrapers/example-site.ts
import { chromium } from 'playwright';
import { extractProducts } from '../extractors/product';
import { saveToCSV } from '../pipelines/store';

export async function scrape(url: string) {
  const browser = await chromium.launch();
  const page = await browser.newPage();
  await page.goto(url);

  const items = await extractProducts(page);
  await saveToCSV(items, 'data/products.csv');

  await browser.close();
}
```

## Getting Started

1. `npm init -y`
2. `npm add playwright`
3. `npx playwright install chromium`
4. Create scraper in `scrapers/`
5. `npm run scrape`

## When To Use This

- JavaScript-rendered pages (SPAs)
- Need browser automation
- Complex interactions (login, clicks)
- Screenshot and PDF generation
- E2E testing doubles as scraping

## Trade-offs

**`Heavy dependencies`** - Browser binaries are large

**`Slower than HTTP`** - Full browser is slower than raw requests

**`Resource intensive`** - Each browser uses significant memory

## Best Practices

- Use headless mode in production
- Implement delays between requests
- Handle navigation timeouts gracefully
- Block unnecessary resources (images, fonts)
- Test extractors with saved HTML

## Naming Conventions

**`Scrapers`** - Named by target site (example-site.ts)

**`Extractors`** - Named by data type (product.ts, listing.ts)

**`Types`** - Interfaces for scraped items