Decisions and tradeoffs

* Use postgres for persistent storage instead of in memory or file based storage to enable versioning and querying
* Chose Puppeteer over selenium to for more modern and headless webcrawling
* Modified links to route through the backend server instead of relative paths so links within archived pages function correctly
* Enabled dynamic port allocation to avoid frontend backend conflicts on localhost
* Modified webcrawler to track all pages and local file paths, added updateInternalLinks method that runs after all pages are crawled. Created url to file path mappings for link resolution.
* Implemented dynamic query of database for all pages in an archive when serving html
* Added gzip compression for html and css files
* Hosting backend on railway for simplicity to deploy postgres and backend

Tradeoff compatibility vs implementation complexity

* Some complex websites with Javascript heavy content may not fully load all assets
* Cheerio-based html parsing and link rewriting system that works for traditional websites, but sometimes fails to load some dynamic content
* Followed 80/20 rule, most websites work well, mvp focus

What I would do differently with more time

* Improve asset loading with cheerio for certain websites that have complex dependencies.
* For future archives I would only archive new information
* Store assets on s3 instead of postgres for better scalability
* Add multi threading or worker queues for better crawling speed
* Refactor crawler to only store deltas(new information) for repeat archives

How you would scale this solution for production use

* Move storage to s3 with compression and deduplication for efficiency
* Introduce microservices and worker queues for crawling, archiving, and serving snapshots
* Add monitoring and health checks, rate limiting, and retry logics for failed crawls
* Scale horizontally with docker containers for distributed crawling
* Use Redis for caching metadata and improving snapshot load speed