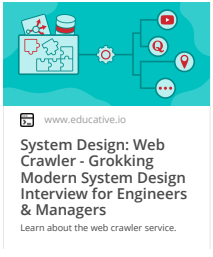
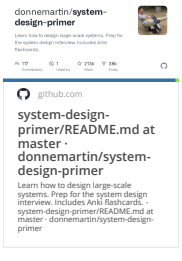




Step 1 - Understand the problem and establish design scope

The basic algorithm of a web crawler is simple:

1. Given a set of URLs, download all the web pages addressed by the URLs.
2. Extract URLs from these web pages
3. Add new URLs to the list of URLs to be downloaded. Repeat these 3 steps.

Does a web crawler work truly as simple as this basic algorithm? Not exactly. Designing a vastly scalable web crawler is an extremely complex task. It is unlikely for anyone to design a massive web crawler within the interview duration. Before jumping into the design, we must ask questions to understand the requirements and establish design scope:

Candidate: What is the main purpose of the crawler? Is it used for search engine indexing, data mining, or something else?
Interviewer: Search engine indexing.

Candidate: How many web pages does the web crawler collect per month?
Interviewer: 1 billion pages.

Candidate: What content types are included? HTML only or other content types such as PDFs and images as well?
Interviewer: HTML only.

Candidate: Shall we consider newly added or edited web pages?
Interviewer: Yes, we should consider the newly added or edited web pages.

Candidate: Do we need to store HTML pages crawled from the web?
Interviewer: Yes, up to 5 years

Candidate: How do we handle web pages with duplicate content?
Interviewer: Pages with duplicate content should be ignored.

Above are some of the sample questions that you can ask your interviewer. It is important to understand the requirements and clarify ambiguities. Even if you are asked to design a straightforward product like a web crawler, you and your interviewer might not have the same assumptions.

Beside functionalities to clarify with your interviewer, it is also important to note down the following characteristics of a good web crawler:

- Scalability: The web is very large. There are billions of web pages out there. Web crawling should be extremely efficient using parallelization.
- Robustness: The web is full of traps. Bad HTML, unresponsive servers, crashes, malicious links, etc. are all common. The crawler must handle all those edge cases.
- Politeness: The crawler should not make too many requests to a website within a short time interval.
- Extensibility: The system is flexible so that minimal changes are needed to support new content types. For example, if we want to crawl image files in the future, we should not need to redesign the entire system.

Для чего?

Для кого?

Соотношение чтение: запись

Проблемы:

Какой изначальный набор ссылок?

Анализ ссылок

мы гугл? да

ТБ данных, миллиарды страниц

Надо хранить много данных

Что нам делать с динамическим контентом?

Get запрос (Js based не работает)

Runtime эмуляция (поднять браузер)

Работа с контентов

нет

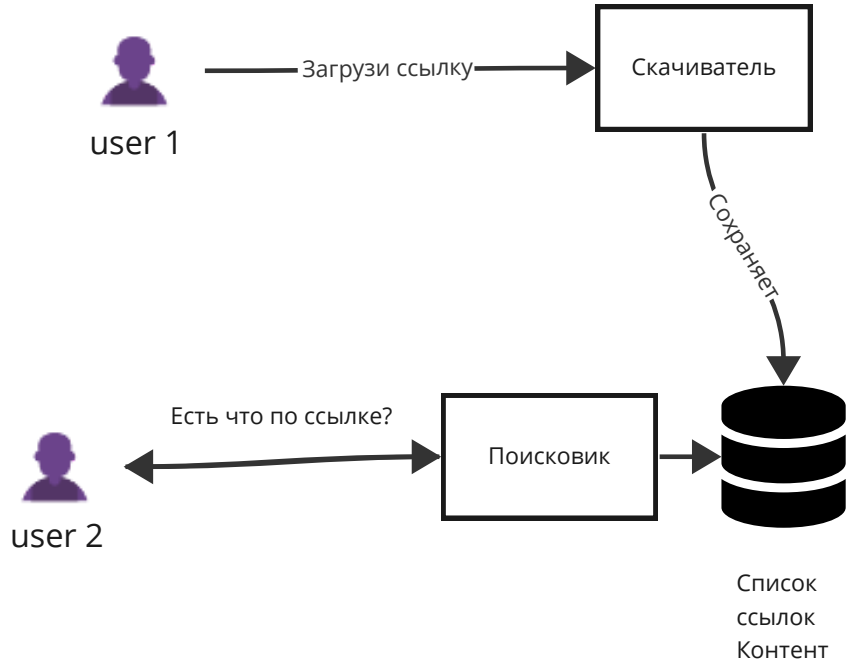
ГБ данных, ограниченный набор страниц

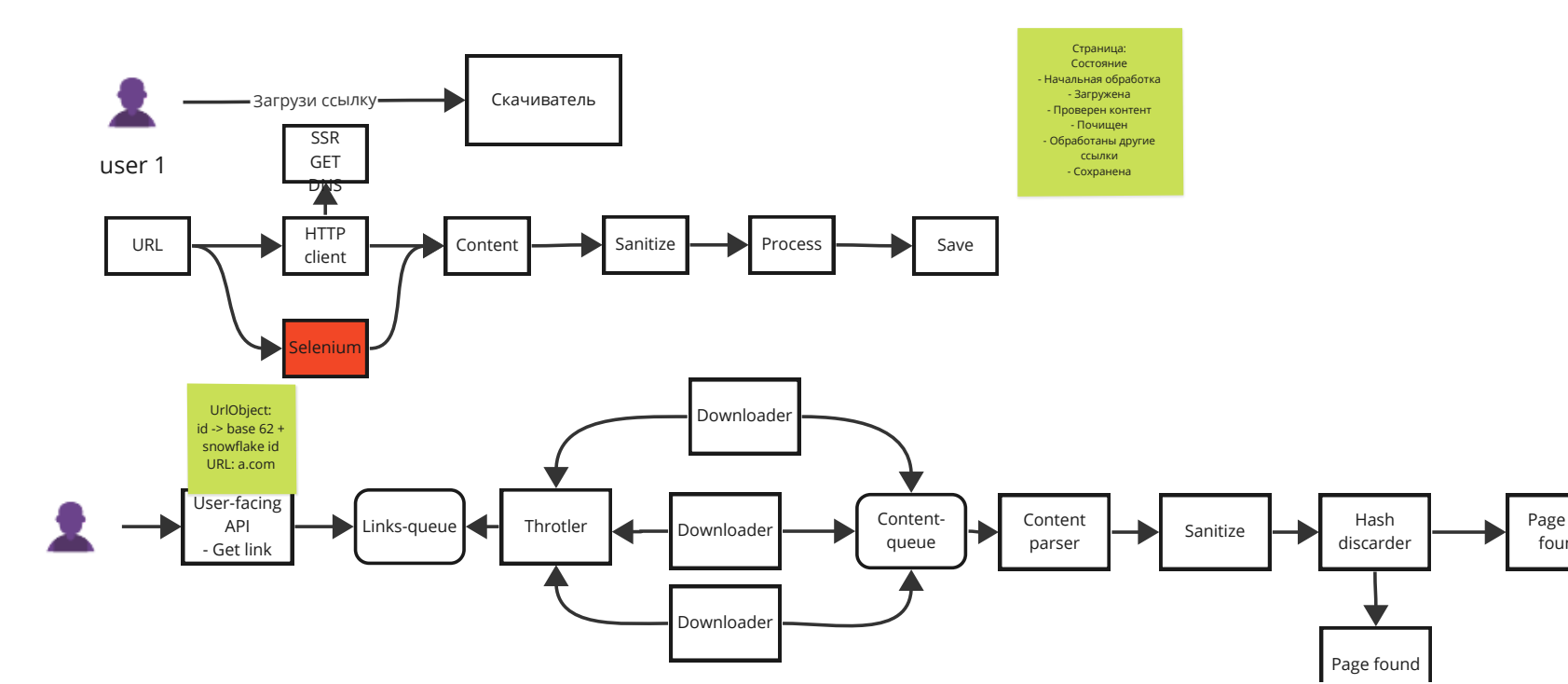
Как будем хранить дубликаты?

Что делаем с картинками и тд?

API

- **Загрузить ссылку для анализа**
- **Анализируем контент**
- **Ищем другие ссылки**
- **Запускаемся снова**
- **Получить контент по ссылке**

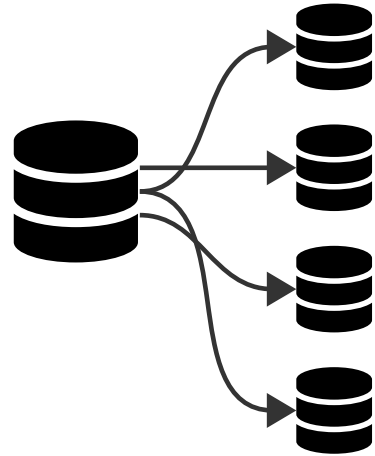



Загрузи ссылку
Скачиватель
user 1
SSR
GET
DNS
URL
HTTP client
Content
Sanitize
Process
Save
Selenium
Страница:
Состояние
- Начальная обработка
- Загружена
- Проверен контент
- Почищен
- Обработаны другие ссылки
- Сохранена
UrlObject:
id -> base 62 + snowflake id
URL: a.com
User-facing API
- Get link
Links-queue
Throtler
Downloader
Downloader
Downloader
Content-queue
Content parser
Sanitize
Hash discarder
Page found

Загрузи ссылку

Скачиватель

user 1



F(url) = long

Hash(long) = int

h(L) = L % кол-во шардов

Решардинг

Save

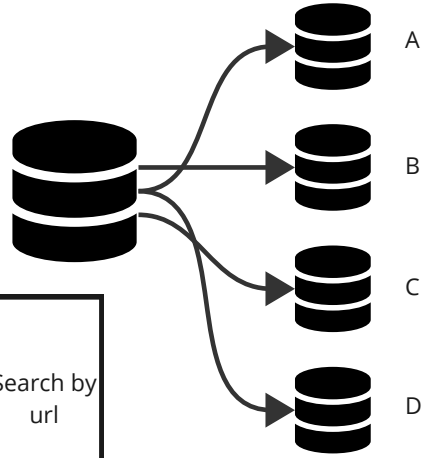
Content (HTML)

Relation

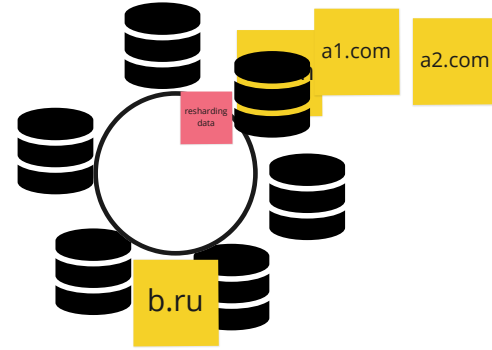
SQL:
- Id
- URL
- PATH
- HASH
- update date

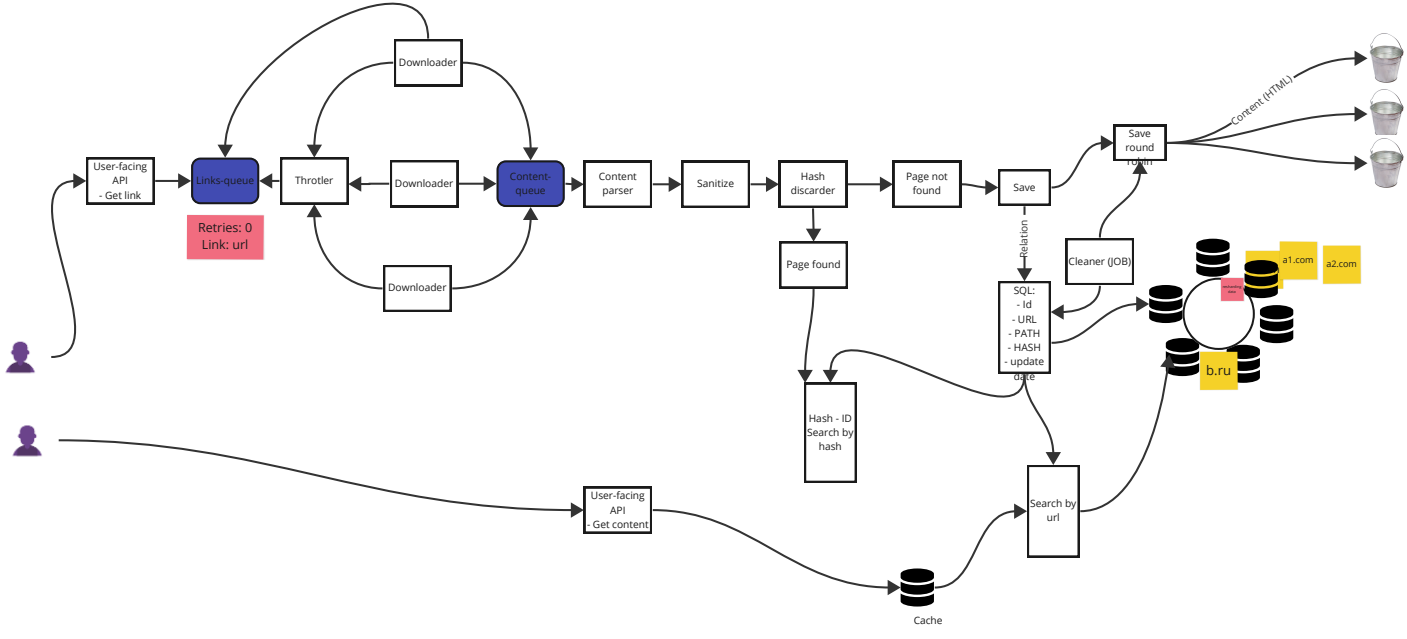
Search by hash

Search by url



23 + 10 = 23 шарда




Downloader
User-facing API - Get link
Links-queue
Throtler
Downloader
Content-queue
Content parser
Sanitize
Hash discarder
Page not found
Save
Save round robin
Content (HTML)
Retries: 0
Link: url
Downloader
Downloader
Page found
Relation
Cleaner (JOB)
SQL:
- Id
- URL
- PATH
- HASH
- update date
a1.com
a2.com
b.ru
Hash - ID
Search by hash
User-facing API - Get content
Search by url
Cache