* + **Functional requirements:**
    - Our service should provide top 10 terms that begin user’s current query as they type.
    - The suggestions should be sorted based on frequency and recentness of phases or queries. This means that most commonly used and recently used terms that match the user’s input will be prioritized in the list of suggestions.
  + **Non-functional requirements:**
    - High-availability: The system should be able to handle disruptions and continue to function without significant downtime.
    - Real-time suggestions: The user should be able to see suggestions in near-instantaneous time, with a target within 200 milliseconds.
    - Scalability: The system should handle a large volume of traffic without experiencing performance issues
  + **Scale/estimation:**
    - 1 trillion pages
    - 100 billion unique pages
    - Each updated once every 10 days
    - Average page is 2MB
    - Blob storage size : 100Billion pages \* 2MB/page= 200 Petabites
  + **Database storage size:**
    - URL: 50B
    - BLOB ID:16 B
    - Title:60B
    - Description:150B
    - Hash:16B
    - Last\_updated:8B
    - Priority:4B
    - 100 Billion pages \* 314Bytes/pages=31.4 TB

* + **API :**

1) getquery

* + GET: /api/v1/getquery
  + Parameters : query: string
    - : Page (for pagination)
  + Return : List<HTML Page> : title, description, URL

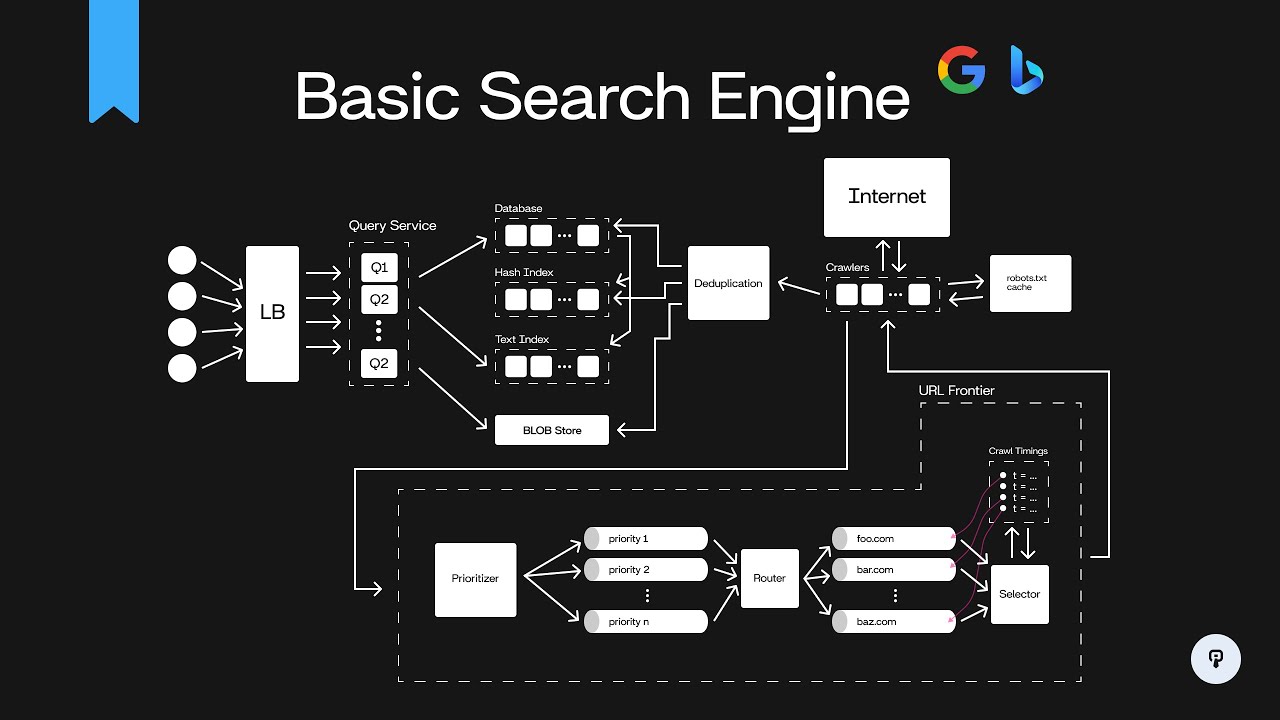
* + **Data model:**

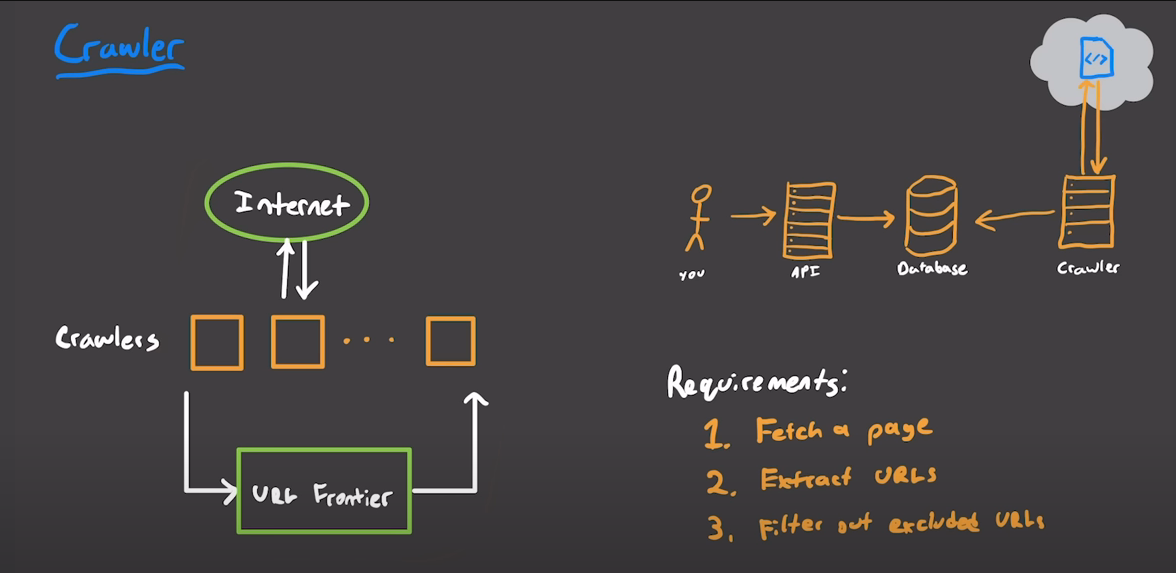
Table Schema :

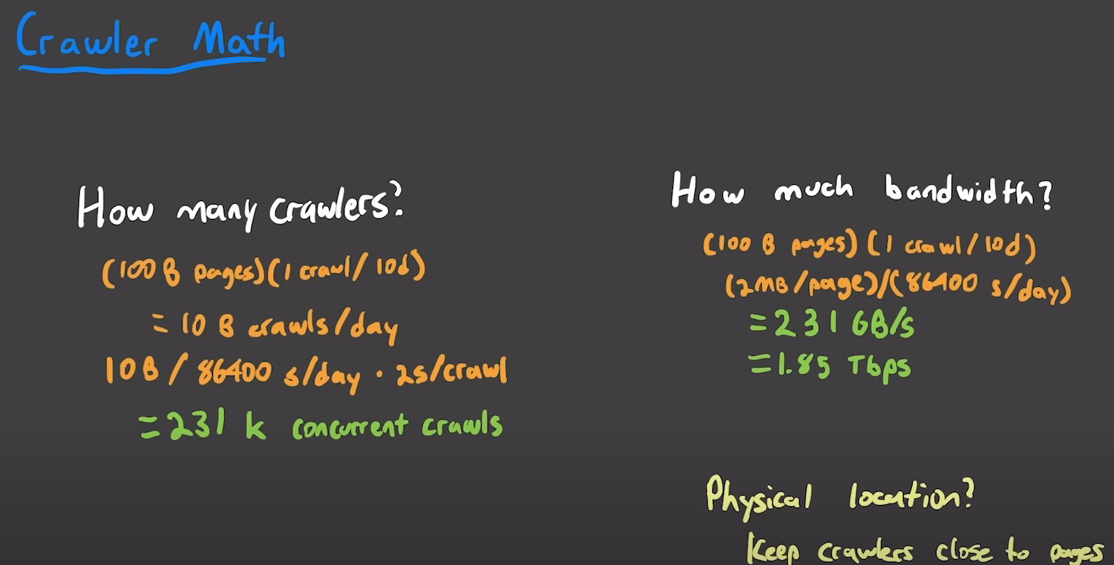
* + Url
  + Blob ID - reference to blob object in Amazon S3 bucket
  + Site content
  + Title
  + Description
  + Hash: since there are 1 trillion pages on internet among which 100 billion pages are unique. Hash is used to store unique pages to avoid large storage of data
  + Last\_updated - how frequently site is being updated
  + Priority: we can use to scrape the pages

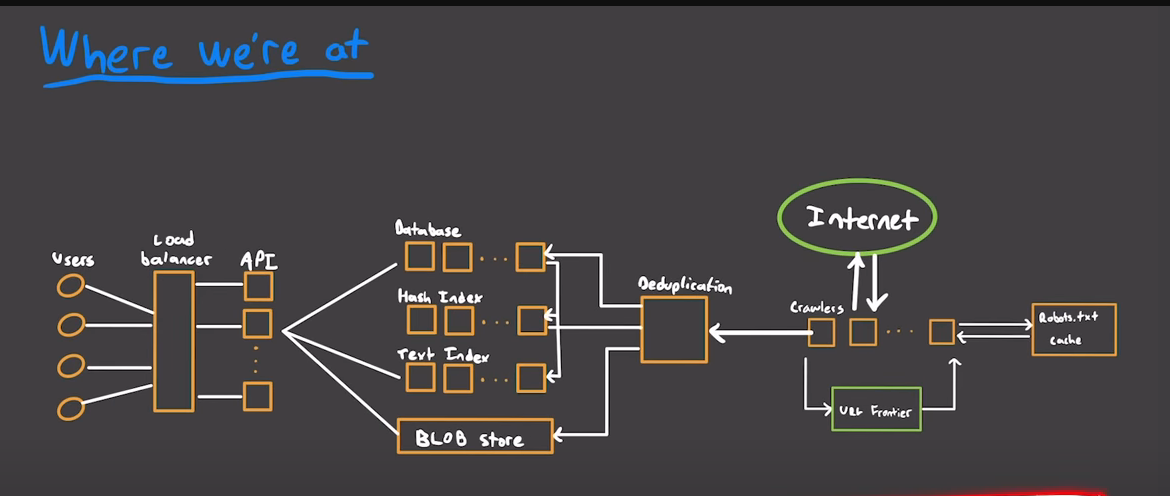
* + **Query patterns:**
    - Get page by url
    - Get page by hash
    - Search for a word

* + **Get page by url:**
    - Database sharding can be used: table is divided into multiple tables and stored in different nodes.
    - Algorithm is run on shard\_key to identify which nodes data will reside.
    - Its very important to make share data has high cardinality and low frequency in order to use shard\_key.
  + **Get page by hash:**
    - We use concept of global index which is itself sharded database.
    - In this case, instead of storing whole data in database we store hash which will act as shard\_key and URL which will be used to lookp data in primary table.
  + **Search for a word :**
    - We use concept of Text Index, it is also sharded database.
    - Database will contain word,freuency and url.
    - We will maintain info on how many times(frequency) and in which URL word is appear.
    - We use word as shard key and sort based on frequency so top most row will be most frequently used word.









* + URL frontier:

