

# Data as a Service

Info 253A: Frontend Web Architecture

Kay Ashaolu

# Why data storage?

- When we make a web request, where do we get the data from?
- When we create data, where do we put it?
- Where do "resources" live?

# Example: bit.ly

- Lots of data to store
  - Shortcut to url mapping
  - Statistics about links
  - Information about users

# Example: bit.ly

```
long url http://news.google.com  
short url http://bit.ly/awekl  
hit count 482240
```

```
long url http://facebook.com/user/profile  
short url http://bit.ly/czasw  
hit count 11023
```

```
long url http://msnbc.com/news/article/  
short url http://bit.ly/olkjpl  
hit count 1232
```

# Data Storage Design

- What is the storage format?
- How do we lay out data?
- How do we access data?

# Why use a file?

- <http://news.google.com>, <http://bit.ly/awekl>, 482240
- <http://facebook.com/user/profile>, <http://bit.ly/czasw>, 11023
- <http://msnbc.com/news/article>, <http://bit.ly/olkjpl>, 1232
- What are the pros and cons?

# Problems with Files

- What if we want to add another field?
- What if we want to query different parts of data? How efficient is this?
- What if we have concurrent accesses?
- What data structures should we use?

# Data Independence

- Databases: apps shouldn't have to worry about these problems!
- Underlying storage format independent of application-level logic



# Relational Data Stores

- RDBMS: Relational Database Management System
- Invented in the 1970s
- e.g., Oracle, MySQL, Postgres, IBM DB2, Microsoft SQL Server

# Relational Model

- Reason about sets of facts, or "tables"
- Each fact is a "row"
- Attributes are "columns" of row

# NoSQL

- Different approach to data storage
- Simple but predictable data models
- Often have to build own features
- Designed for massive scale-out

# Key-Value Store

```
1 put(key, value);  
2 let value = get(key);
```

## Pros

- Simple API
- Easy to understand performance
- Easy to scale and use

## Cons

- Simple API
- Must handle own schema management
- May need to manually implement search features

# Document Store

```
1 {  
2     "long_url": "http://www.google.com",  
3     "short_url": "qwelmw",  
4     "hit_count": 2  
5 }
```

- No predefined schema
- Store handles layout of arbitrary fields
- Examples: MongoDB, CouchDB, Cassandra, Redis

# Front End

- Since we are on the front end, we don't normally deal with the database details
- What we really need is the ability to reach out to a service that gives us the capabilities of using a database
- Sounds like what we need is an API!

# Google Firebase

- A NoSQL Cloud database that we can directly use with our React Applications
- Provides an API to save JSON data to the cloud
- Gives us the ability to save, access, and search data outside of the confines of our application
- We can write our front end code in React, and use API's to provide the functionality we need without a webserver

Questions