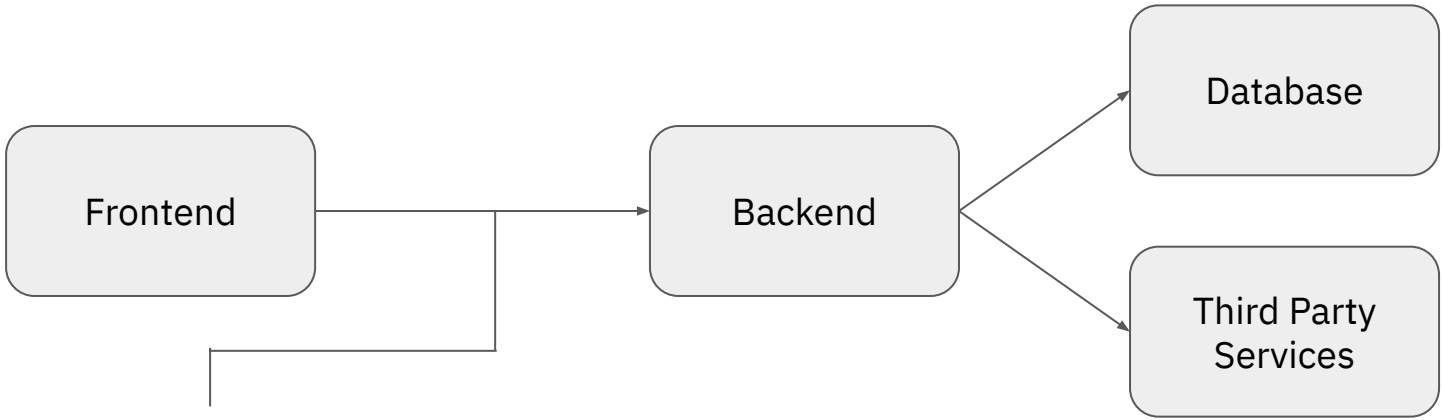
Basics of Web Apps

CS 40 | January 10, 2024

Agenda

1. Frontend
2. Backend
3. Reasons for Frontend/Backend Split
4. Transport Method
5. Databases (briefly)
6. Misc (mostly PaaS)

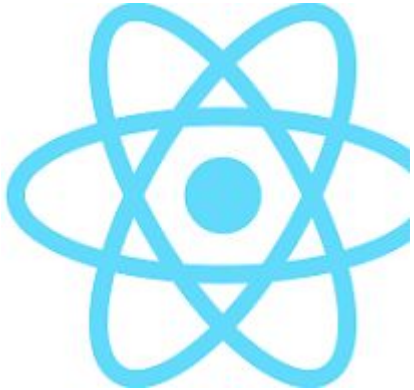
The Basic Architecture of a Web App



- REST
- GraphQL
- Web Socket
- etc

Frontend

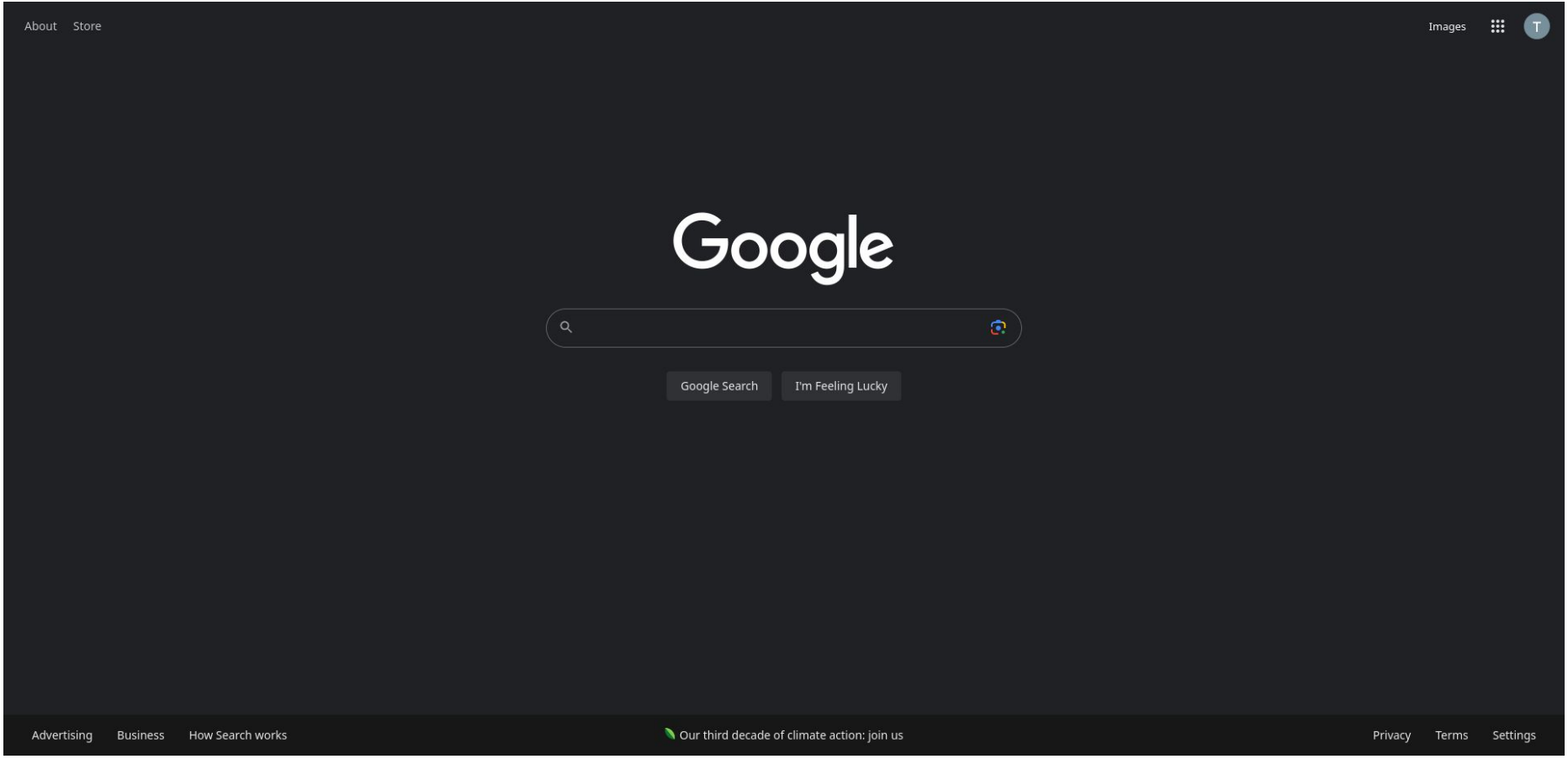
Frontend



- The part the user interacts with
  - Run on the user's device

- Mobile App, Website, etc
  - Web apps: written in one of many frameworks that compiles to Javascript
  - Mobile Apps: written using platform specific libraries or are web apps

- Communicates with the backend over some protocol
  - Communicate "state" between backend and frontend


About
Store
Images
T
Google
Google Search
I'm Feeling Lucky
Advertising
Business
How Search works
Our third decade of climate action: join us
Privacy
Terms
Settings

CS 40 @ Stanford

Cloud Infrastructure and Scalable Application Deployment

Winter 2024

Mondays & Wednesdays, 4:30 PM – 5:50 PM
530-127 (enter from Duena St or Building 530 courtyard)

Trying to launch your next viral programming project and anticipating substantial user growth? This course will help you learn to implement your ideas in the cloud in a scalable, cost-effective manner. Topics will include cloud AI/ML pipelines, virtual machines, containers, basic networking, expressing infrastructure as code (IaC), data management, security and observability, and continuous integration and deployment (CI/CD). Through hands-on learning and practical examples, you'll learn to effectively deploy and manage cloud infrastructure. There is no out-of-pocket cost associated with this class and cloud credits will be provided for all students.

Course Staff

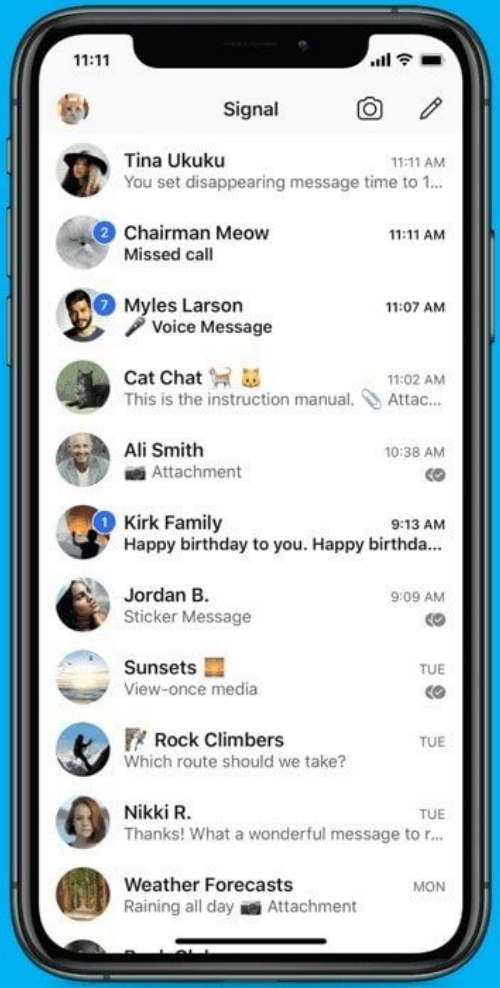
Aditya Saligrama
Instructor
saligrama@stanford.edu

Cody Ho
Instructor
codyho@stanford.edu

Ben Tripp
Teaching Assistant
btripp@stanford.edu

Mike Abbott
Advisor

Christos Kozyrakis
Faculty Sponsor


11:11
Signal
Tina Ukuku
11:11 AM
You set disappearing message time to 1...
Chairman Meow
11:11 AM
Missed call
Myles Larson
11:07 AM
Voice Message
Cat Chat
11:02 AM
This is the instruction manual. Attac...
Ali Smith
10:38 AM
Attachment
Kirk Family
9:13 AM
Happy birthday to you. Happy birthda...
Jordan B.
9:09 AM
Sticker Message
Sunsets
TUE
View-once media
Rock Climbers
TUE
Which route should we take?
Nikki R.
TUE
Thanks! What a wonderful message to r...
Weather Forecasts
MON
Raining all day Attachment

The Idea of State

- State: in general, all the information a website stores
  - User
    - All authentication information
  - Images/Video
  - Contacts
  - Everything else

- Frontend is responsible for displaying the state to the user in the app

Backend

Backend

- Contains all the application logic
    - Makes frontend smaller (and therefore faster, easier to load, and consume less bandwidth)
- Communicates with external resources (including the database)
- Runs on a server you own
- Presents some API to the frontend

Monolith vs Microservices

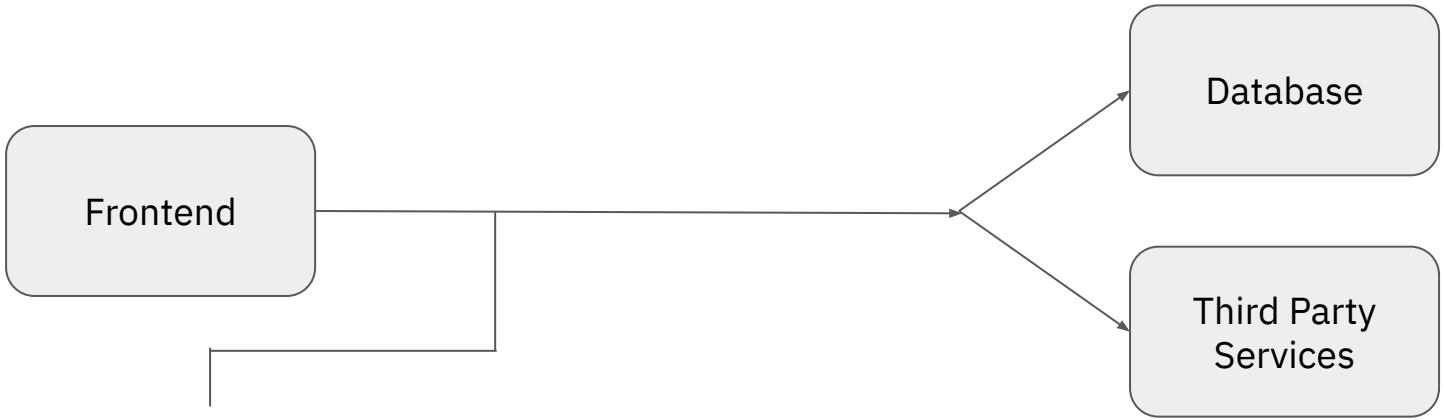
- Modern web applications are very complex with many services
  - Example: `amazon.com` has the product, a payment method, reviews, comments, recommendations, etc

- How can we structure a backend for reliability and scalability?

- In general:
  - Start building applications as monoliths composed of various services
  - If needed, for scaling/reliability reasons, break off services into separate microservices

Why do we need a backend?

Why can't we do this (or can we)?



- REST
- GraphQL
- Websocket
- gRPC
- etc.

A: The frontend is *untrusted*

Do *NOT* serve any secrets to the frontend

Why backend?

- Security: hide API keys and other secrets
- Simplifies the frontend
- Flexibility: can push changes to the frontend and backend separately as long as the API is not changed

Transport Protocols

Transport Protocols

- RE(presentative) S(tate) T(ransfer)
- Web sockets
- GraphQL

REST

- Backend hosts various endpoints
  - Examples:
    - `/api/user/register/`
    - `/api/user/login/`
    - `/api/user/me/`

- Frontend makes queries to backend api endpoints

- REST is probably what you should use if you're building a web app

Web Sockets

- Maintain a continuous connection with the backend, all communication occurs over this connection

- Use for continuous data transfers

GraphQL

- Idea: Abstract over the backend, query data using a custom syntax
- Built by Meta in 2015 to deal with the demands of high scaling
- Works best only for very large and distributed web apps

```
query {
  authors {
    totalCount
    edges {
      cursor
      node {
        id
        _id
        firstName
        lastName
        quotes {
          edges {
            node {
              id
              _id
              quote
            }
          }
        }
      }
    }
    pageInfo {
      startCursor
      hasNextPage
      hasPreviousPage
      endCursor
    }
  }
}
```

Databases

Databases

- Types (massively oversimplified)
  - SQL
  - NoSQL
  - Vector
  - In-memory (Redis)

SQL Databases

- Idea: store everything in tables
- Multiple different SQL databases:
  - PostgreSQL
  - MySQL
  - MSSQL
- S(tructured) Q(uery) L(anguage)
  - Language that queries data stored in tables

```
SELECT p.*
FROM Production.Product p
    JOIN Sales.SalesOrderDetail sod ON p.ProductID = sod.ProductID;
```

NoSQL Databases

- Idea: Store data in something that isn't a table
  - Key-Value
  - Tree
  - Many others

- Examples:
  - DynamoDB
  - MongoDB
  - Countless others

Vector Databases

- Idea: store everything as high dimensional vectors
  - Useful for ML tasks
  - Allows you to query based on similarity or difference

- Examples:
  - ChromaDB
  - Pinecone
  - Countless others

In Memory Databases (Redis)

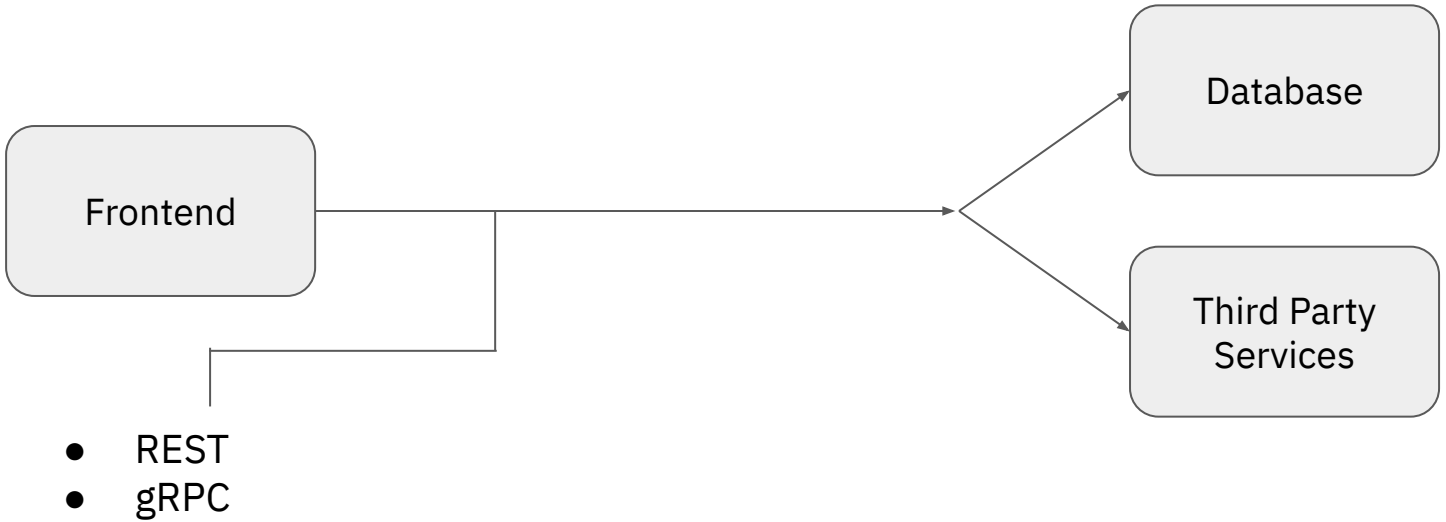
- Idea: Some things need to be very fast, so store it in RAM instead of on disk

- Example uses:
  - Rate limiting and DDoS prevention (slow access times → DDoS is more effective)
  - Accelerating user queries by mapping user IDs to database shards

- Noteworthy providers: Redis, memcached

Misc. Topics

Firebase



Firebase

- PaaS: Platform as a Service
  - Essentially, ignore all the hosting details, only bring the code

- Frontend makes queries directly to database, database contains some backend logic
  - “Backend-as-a-service”

- Applications must be written for the Firebase API

- Incredibly easy to write insecure applications
  - https://mrbruh.com/chattr/
  - https://saligrama.io/blog/post/firebase-insecure-by-default/

Vercel

- Another PaaS service
- Serves a frontend allowing you to ignore hosting details
- Uses AWS below the hood
  - You can save money by using AWS directly
- In general, many parts of the stack can be delegated to a PaaS company

Authentication Providers

- Idea: many websites require a login, and logins are difficult, so have a trusted third party manage login info

- Providers:
  - Google
  - Apple
  - Microsoft
  - Auth0
  - Okta
  - etc.

- Used by https://provisiondns.infracourse.cloud