Australian National University

Full Stack Web Dev Overview

Comp1710/6780 Week 11

Xuanying Zhu
(xuanying.zhu@anu.edu.au)
School of Cybernetics
The Australian National University

Acknowledgement of Country

Outline

About Xuanying Zhu

Currently a lecturer at ANU School of Cybernetics
- Course convenor for CECS8001

PhD at ANU School of Computing
- Associate lecturer for COMP1710/6780

Software Engineer in industry

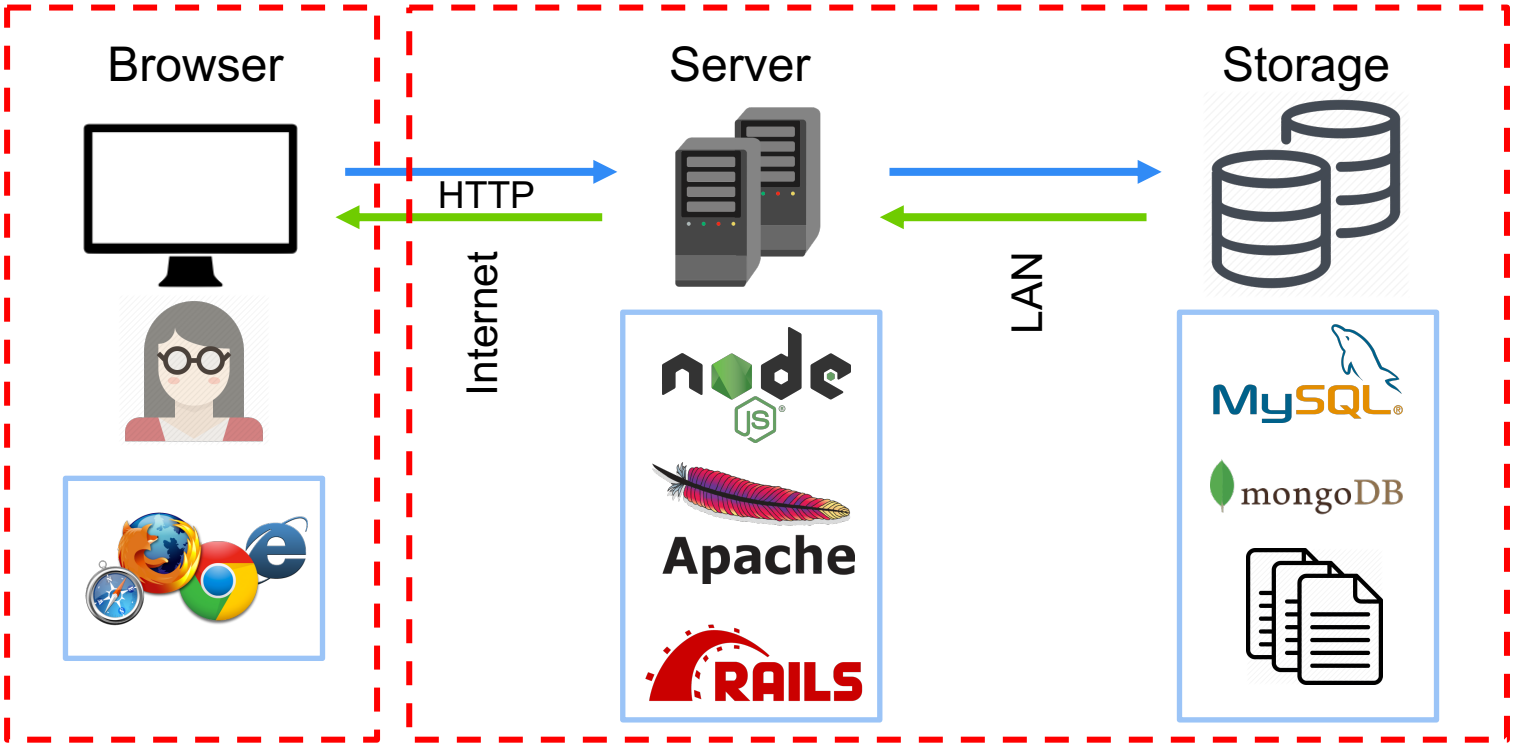
Research Interests: When HCI meets AI/ML

Poll Everywhere

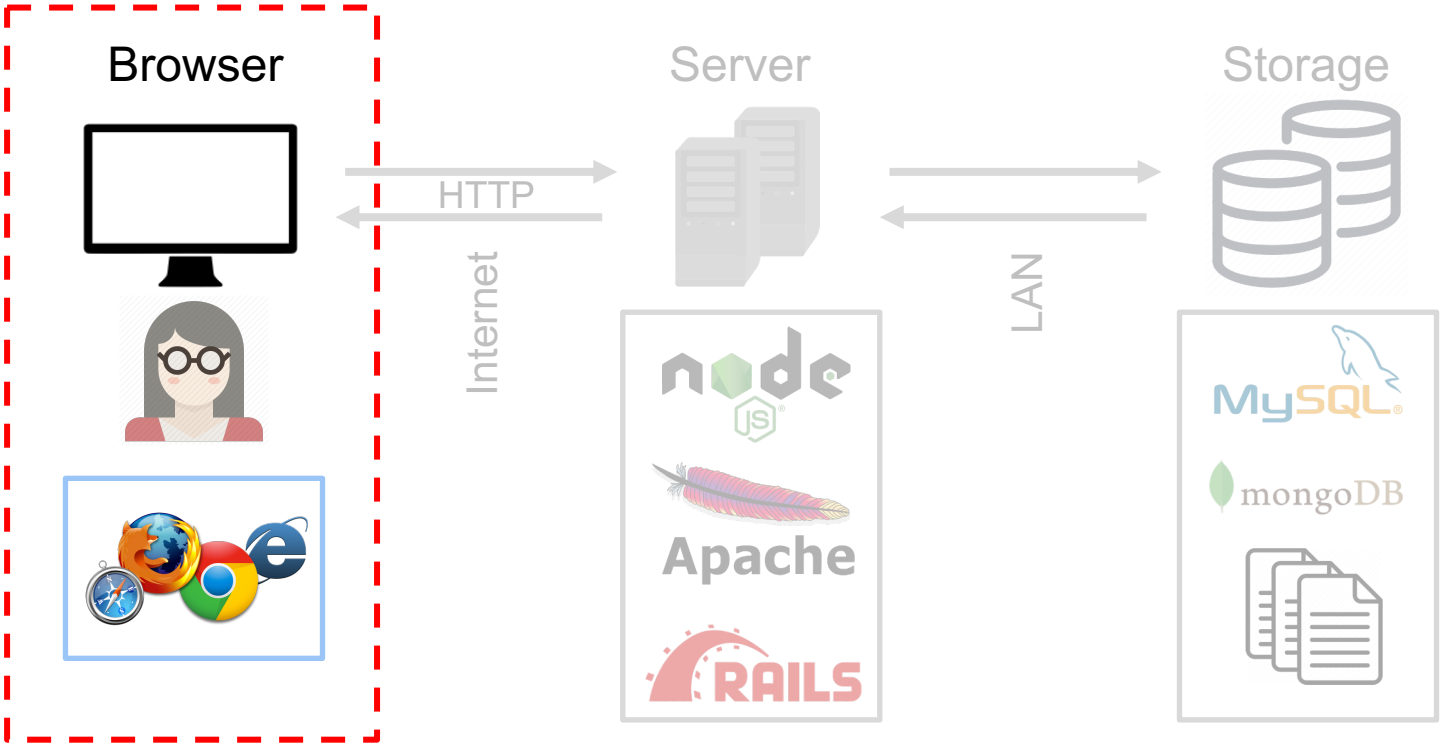
https://pollev.com/xuanyingzhu488

Q1&2: Your understanding of front end

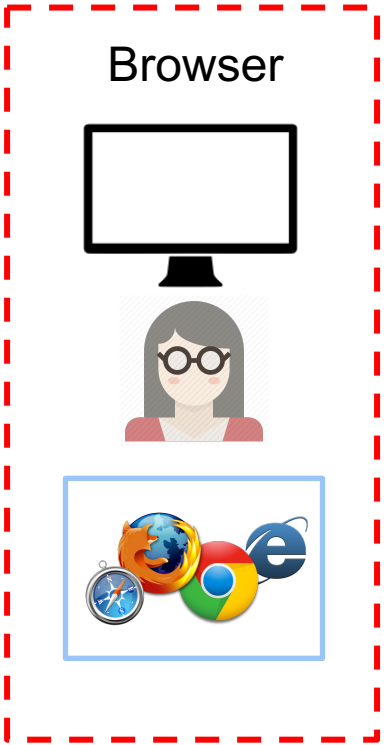
Full Stack Web Application Architecture



Full Stack Web Application Architecture



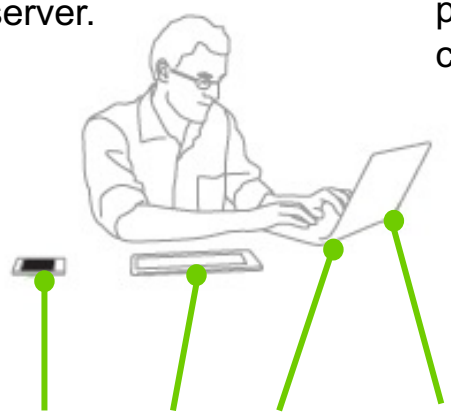
Front End Development



- Also called “Client-side” programming
- It is everything that users see and interact with in the browser.

Front End Development

1 A site is loaded in a browser from the server.

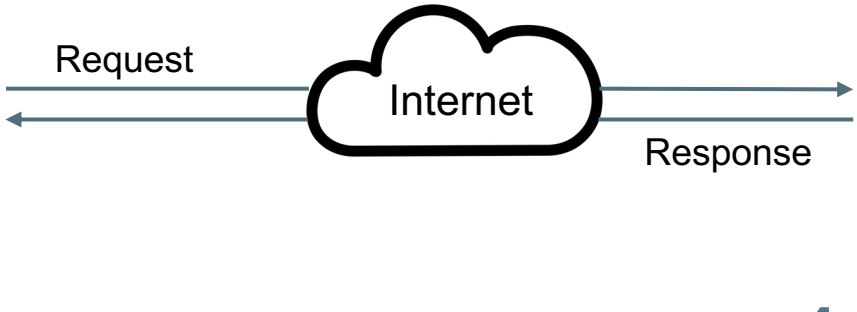


Responsive design allows a site to adapt to different devices.

2 **Client-site scripts** Run in the browser and process requests without call-backs to the server

Everything a user sees in the browser is a mix of **HTML, CSS, and Javascript**.

3 When a call to database is required, scripts send requests to the back end.



4 The **back-end server-side scripts** process the request, pull out data from database and send it back.

Good Web Applications

Design + Implementation

Design

- Good & Bad Design
- Style Guide
  - e.g. Material Design

Implementation

- HTML, CSS, Javascript
- Responsive Design
- CSS frameworks

Good Web Applications

Design

+

Implementation

- Good & Bad Design
- Style Guide
  - e.g. Material Design

- HTML, CSS, Javascript
- Responsive Design
- CSS frameworks

Good Web Applications - Design

Design

- Good & Bad Design
- Style Guide
  e.g. Material Design

Some Design Goals:

- Intuitive to use
  - Don’t need to take a course or read a user manual
- Accomplish task accurately and rapidly
  - Provide needed information and functionality
- Users like the experience
  - Joy rather than pain when using the application

Good Web Applications - Design

Design

Good & Bad Design

Style Guide

e.g. Material Design

Some guiding design principles

- Be consistent
  - Cognitive load less for the user
- Provide context
  - User should not get lost in the app

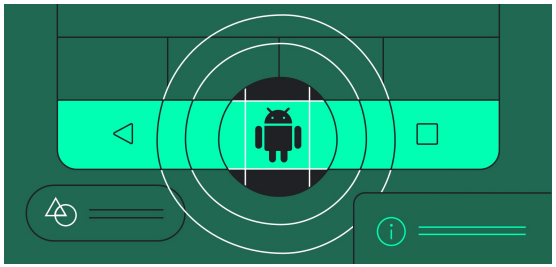
Consistency: Style guides & templates

- A style guide – covers the look and feel
  - Define style, user interactions, layout
- Patterns – do something multiple places in the same way
- Design templates – follow a familiar structure

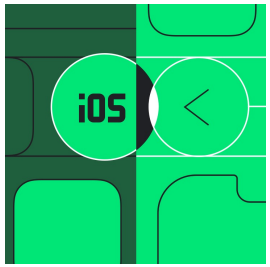
Style Guide Example

Material Design from Google

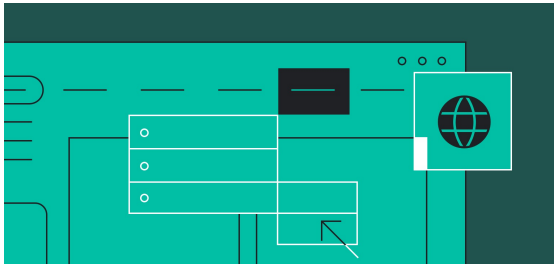
https://material.io/develop/



Android



iOS



Web

Style Guide Example

Material Design from Google

https://material.io/develop/

- Used in Google apps (e.g Android, web apps)
- Two goals:
  1. It unifies Google's numerous products
  2. It unifies Android app interfaces
- Focus on traditional print issues: grids, space, typography, colour etc
- Heavy use of animation to convey action

Style Guide Example

Material Design from Google

https://material.io/develop/

Material Design Foundations

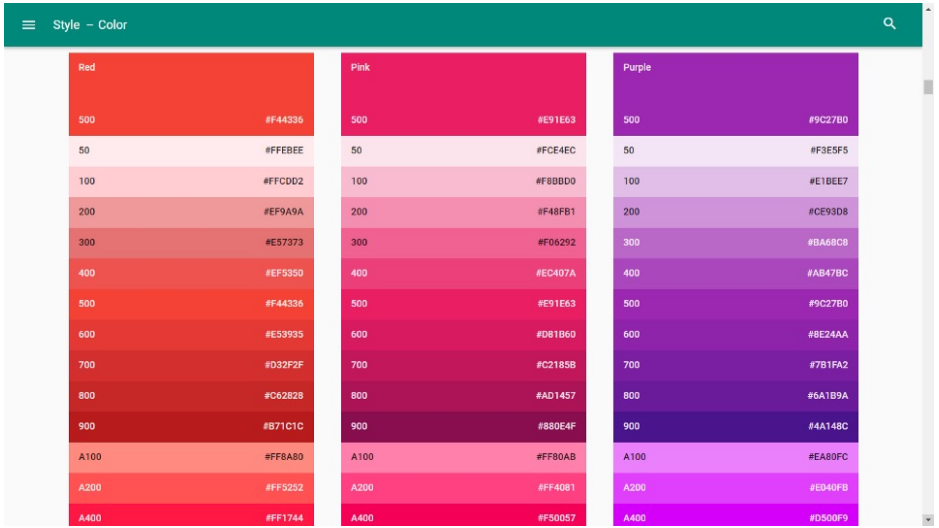
- **Environment** – surfaces, depth, and shadows
- **Layout** – responsive layout grid, white space
- **Navigation** – navigation transitions, search
- **Colour** – suggestions for colours that work well together
- **Typography** – suggestions for point size, weight, spacing
- **Sound** – suggestions for sound attributes
- **Shape** – use shapes to direct attention
- **Motion** – show information, focus attention
- **Interaction** – map touch to actions

Style Guide Example

Material Design from Google

https://material.io/develop/

Colours



Source: Google Material Design guidelines

Style Guide Example

Material Design from Google

https://material.io/develop/

Icons

Source: System Icons collection from Material Design Docs by Google

Style Guide Example

Material Design from Google

https://material.io/develop/

Writing and Typography

Roboto Thin
Roboto Light
Roboto Regular
Roboto Medium
Roboto Bold
Roboto Black
Roboto Thin Italic
Roboto Light Italic
Roboto Italic
Roboto Medium Italic
Roboto Bold Italic
Roboto Black Italic

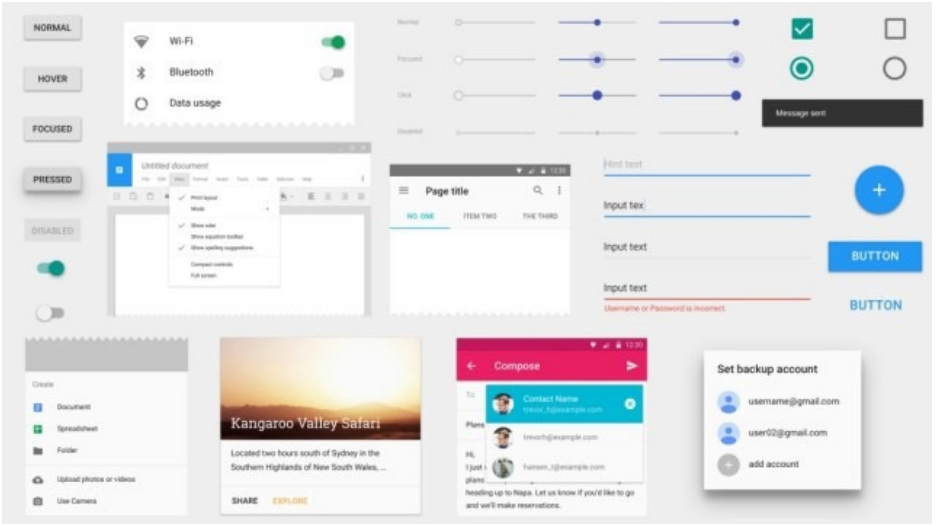
Source: Google Material Design Docs

Style Guide Example

Material Design from Google

https://material.io/develop/

Design Components



Source: Google Material Design Docs

Style Guide Example

Material Design from Google

https://material.io/develop/

How to use it for Web?

Include the material design css and js files. You can make use of cdn or download the files from material design lite

```
<link rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=Material+Icons"
>

<link rel="stylesheet"
      href="https://code.getmdl.io/1.3.0/material.indigo-
pink.min.css">

<script defer
src="https://code.getmdl.io/1.3.0/material.min.js"></script>
```

Good Front-end Applications

Design + **Implementation**

- Good & Bad Design
- Style Guide
  - e.g. Material Design

- HTML, CSS, Javascript
- Responsive Design
- CSS frameworks

Poll Everywhere

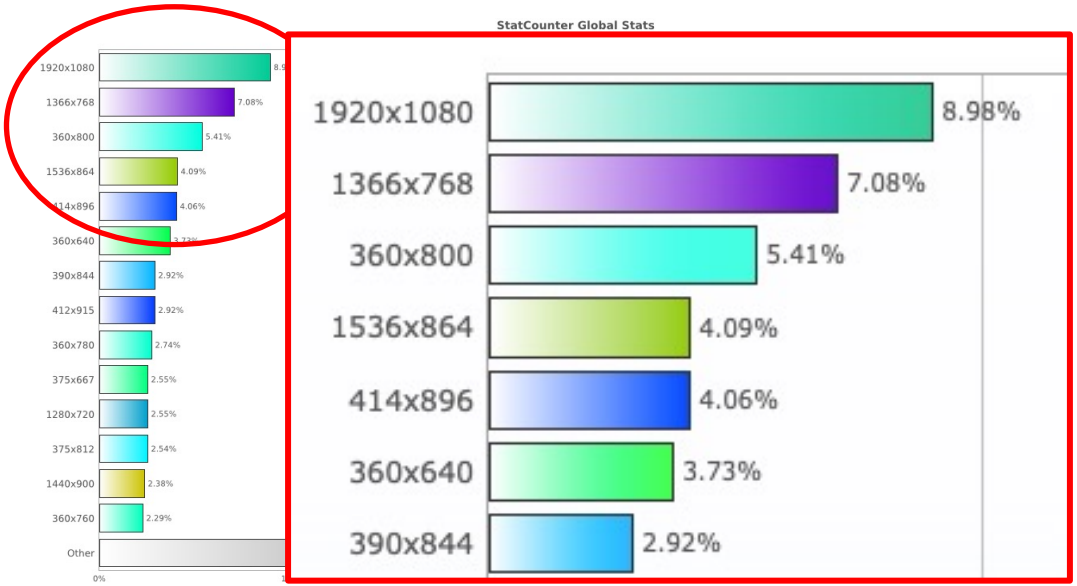
https://pollev.com/xuanyingzhu488

Q3: Your screen size

Responsive Design

Screen Resolution Stats

Top 2: 1920x1080 [8.98%], 1366x768 [7.08%]



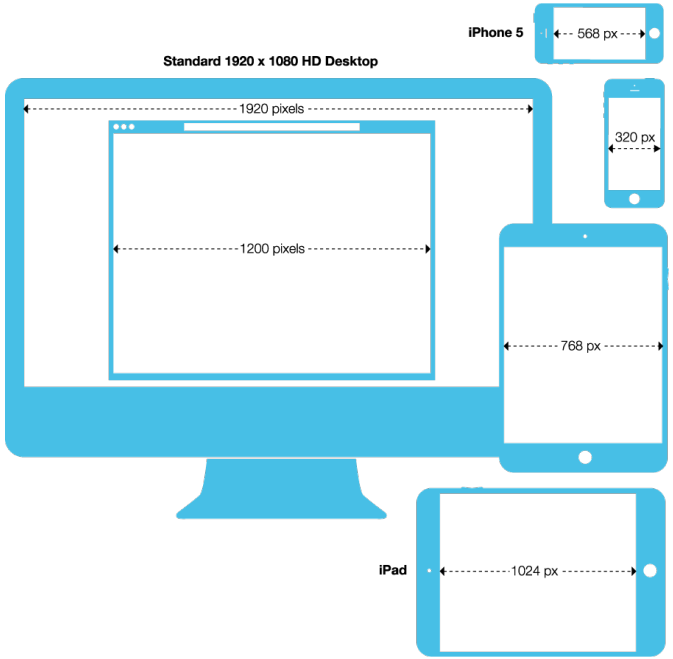
Source: Stat Counter

Implementation

- HTML, CSS, Javascript
- Responsive Design
- CSS frameworks

Responsive Design

Build N versions of each web application?



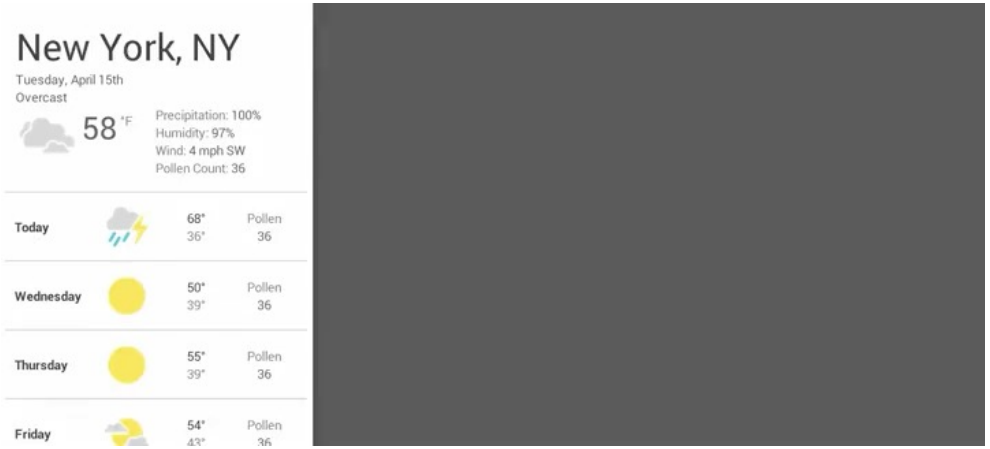
Source: Design Insights

Implementation

- HTML, CSS, Javascript
- Responsive Design
- CSS frameworks

Responsive Design

Example of Responsive Design



Source: Design Insights

Implementation

- HTML, CSS, Javascript
- Responsive Design
- CSS frameworks

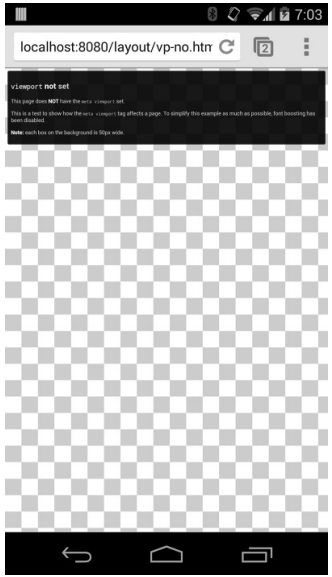
Poll Everywhere

https://pollev.com/xuanyingzhu488

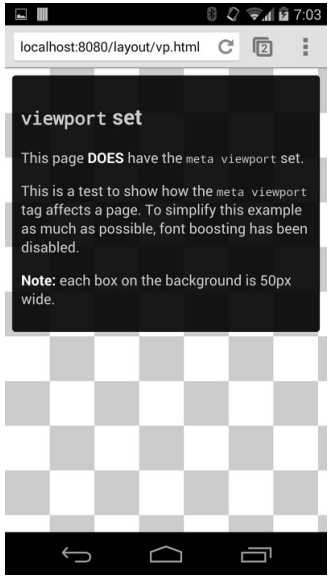
Q4: Your experience with responsive design

Responsive Design

Pages optimized for a variety of devices must include a meta viewport tag in the head.



Page without a viewport set

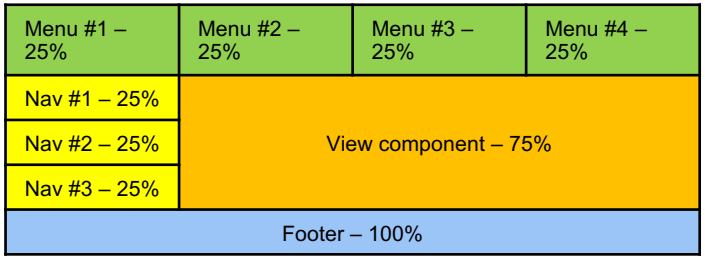


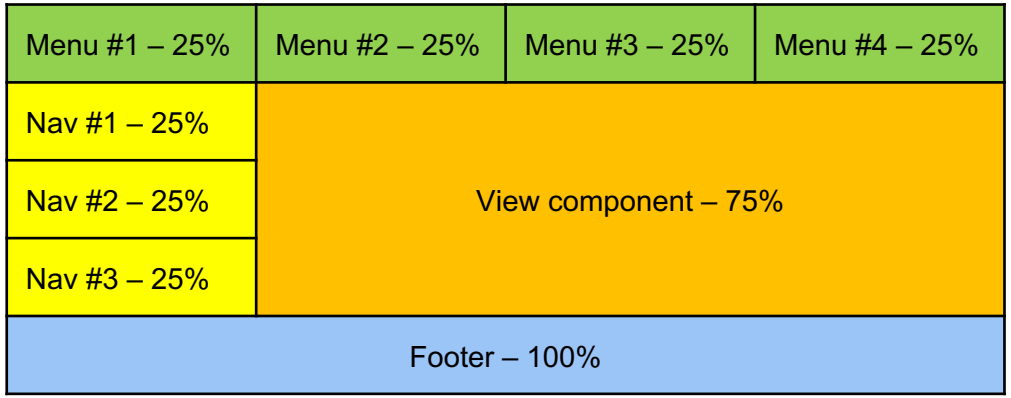
Page with a viewport set

```
<meta name="viewport"
content="width=device-width,
initial-scale=1.0">
```

1 **Set the view port**

Responsive Design






```
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}
```

2 **Add grid layout system with relative (e.g. 50%) rather than absolute (e.g. 50pt) measures**

Responsive Design

Auto-scale images and videos to fit in screen region

Source: Responsive Images

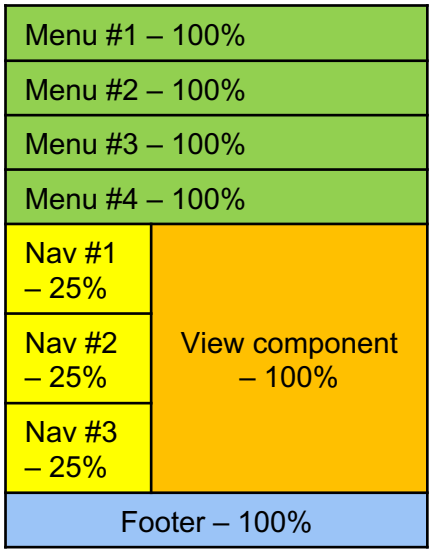
```
img, embed, object, video {
    width: 100%;
    height: auto;
}
```

3 **Make components support relative sizes**

Responsive Design

CSS Breakpoints to control layout

```
@media only screen and (min-width: 768px) {
    /* tablets and desktop layout */
}
@media only screen and (max-width: 767px) {
    /* phones layout */
}
@media only screen and (max-width: 767px) and
(orientation: portrait) {
    /* portrait phones layout */
}
```



4 **Add @media rules based on screen sizes**

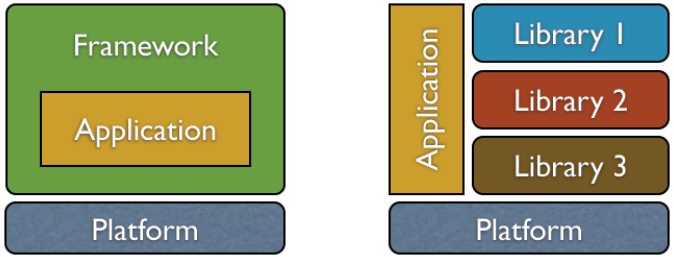
CSS Frameworks

A framework is a standardized set of concepts, practices and criteria for dealing with a common type of problem.

Difference

Libraries: You are in control of when the library should perform a particular function.

Frameworks: The control flow is in the framework and you can customize it.



Source: Tom Lokhorst's blog

Implementation

- ☑ HTML, CSS, Javascript
- ☑ Responsive Design
- ➡ CSS frameworks

CSS Frameworks

Advantages

- Easier code maintenance
- Coherent organizational structure
- Responsive media queries
- Uniform styling across buttons, forms etc.
- Consistent set of fonts and icons

Source: troxler's github

Implementation

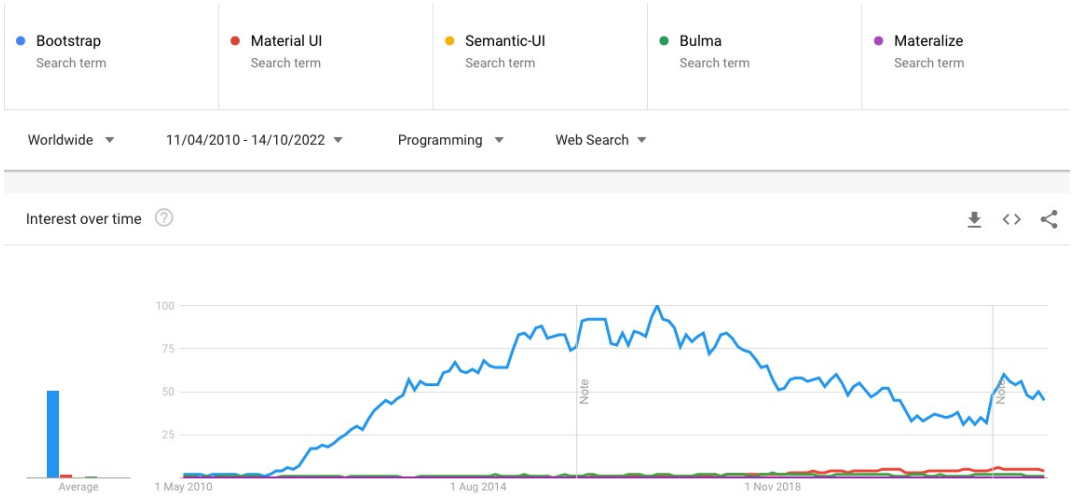
- ☑ HTML, CSS, Javascript
- ☑ Responsive Design
- ➡ CSS frameworks

Poll Everywhere

https://pollev.com/xuanyingzhu488

Q5: Your understanding of front-end/css frameworks

CSS Frameworks



Source: Google Trends

CSS Frameworks GitHub stars

- Bootstrap: 160K Stars
- Material UI: 82K Stars
- Semantic-UI: 50.2K Stars
- Bulma: 46.5K Stars
- Materialize: 38.7K Stars

CSS Frameworks

What are the best CSS frameworks to learn?

Depends on your skills. Having a solid understanding of HTML, CSS, and JavaScript is still the most important skill overall.

Bootstrap
https://getbootstrap.com/

Materialize
https://materializecss.com/

Bulma
https://bulma.io/

Semantic UI
https://semantic-ui.com/

Material UI
https://material-ui.com/

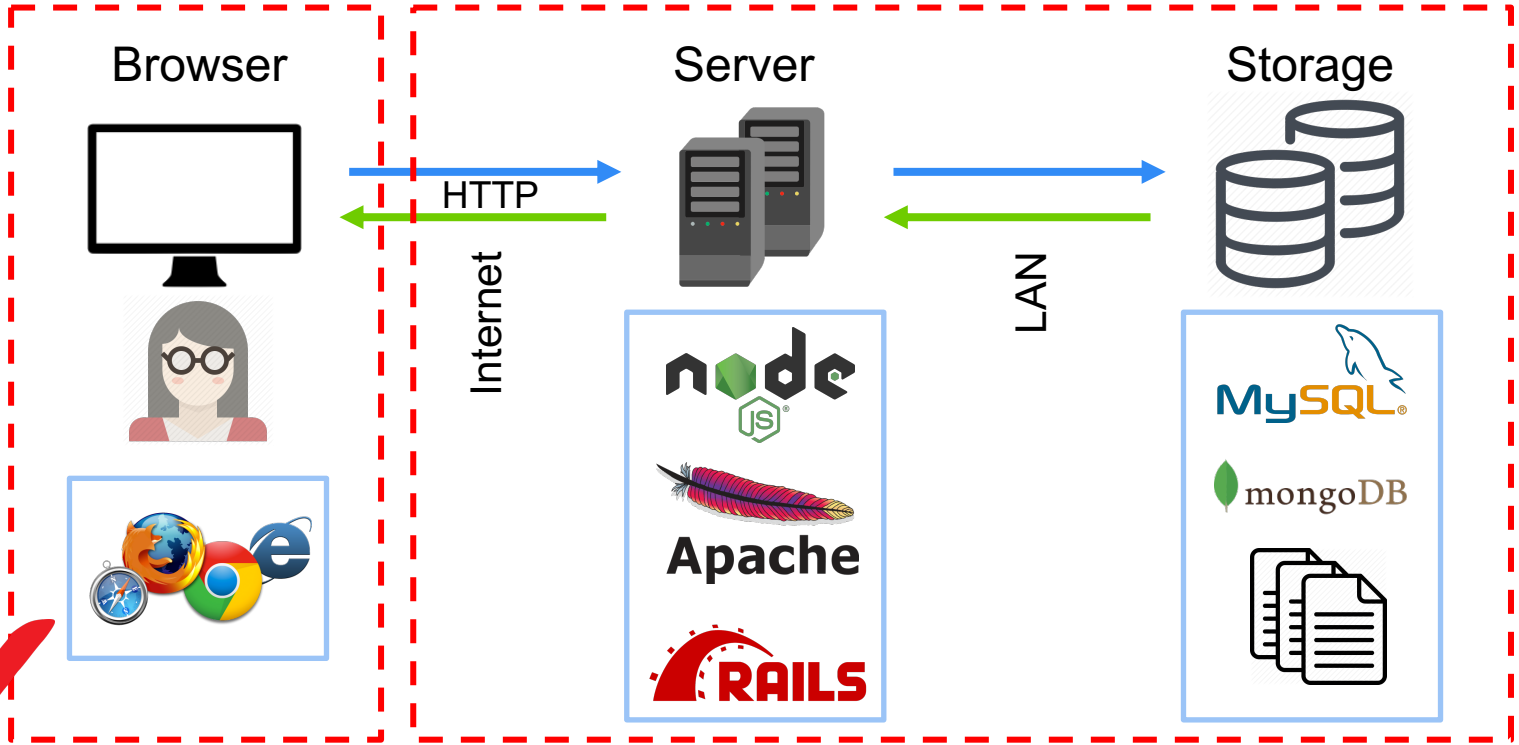
Good Front-end Applications

Design + **Implementation**

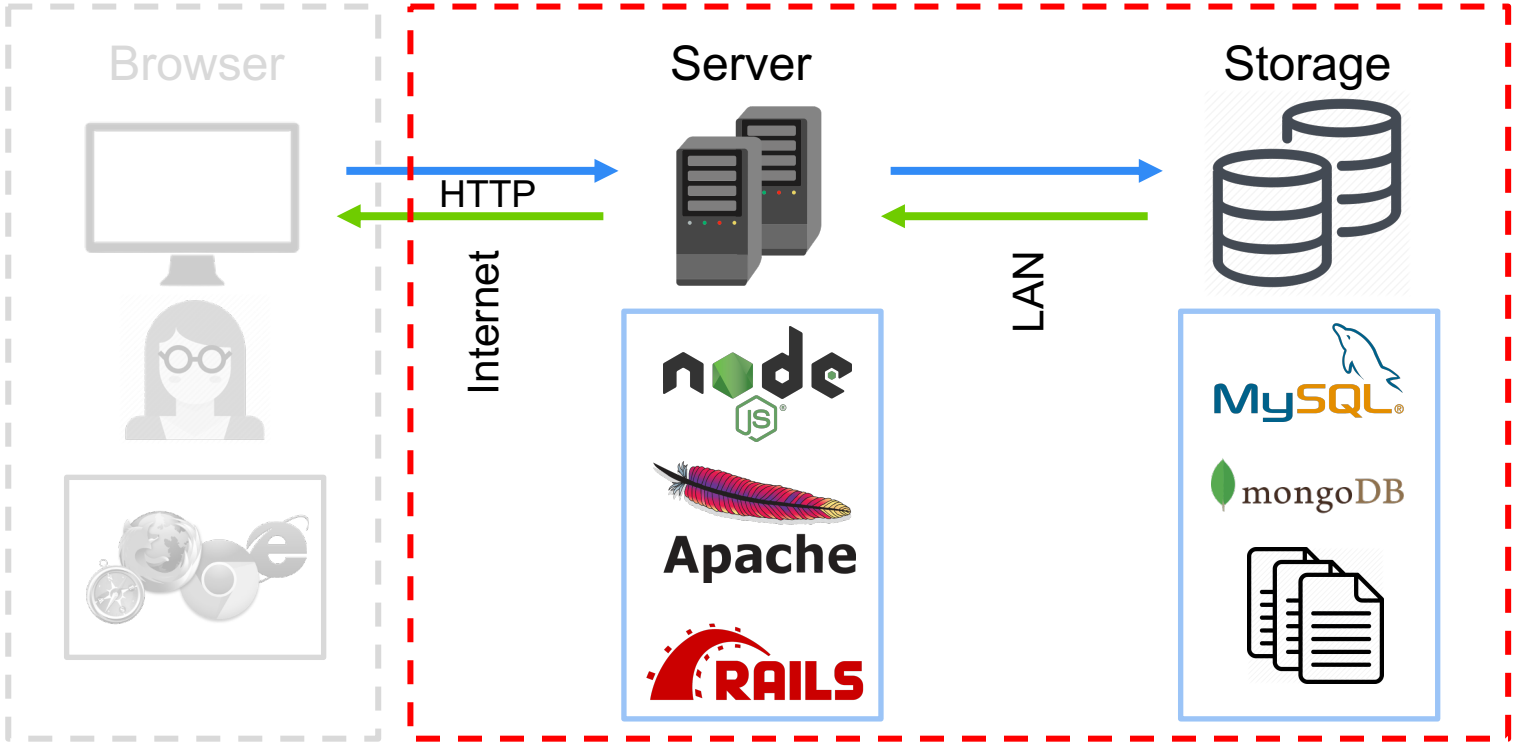
- Good & Bad Design
- Style Guide
  - e.g. Material Design

- HTML, CSS, Javascript
- Responsive Design
- CSS frameworks

Full Stack Web Application Architecture



Full Stack Web Application Architecture



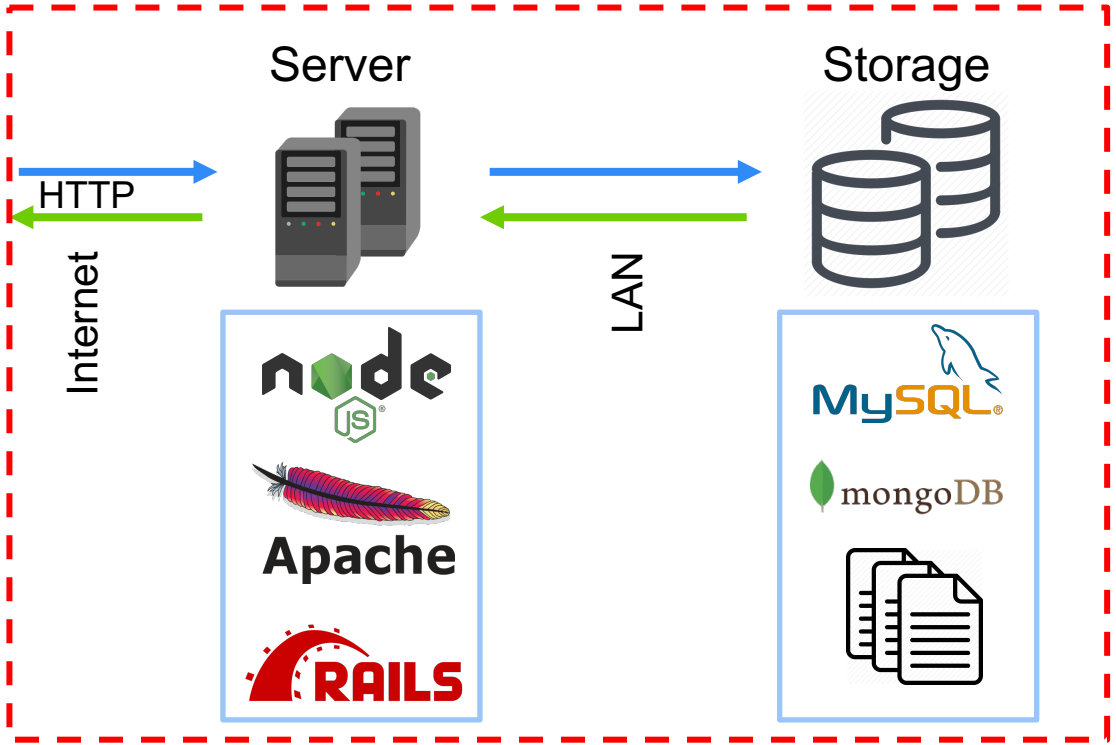
Poll Everywhere

https://pollev.com/xuanyingzhu488

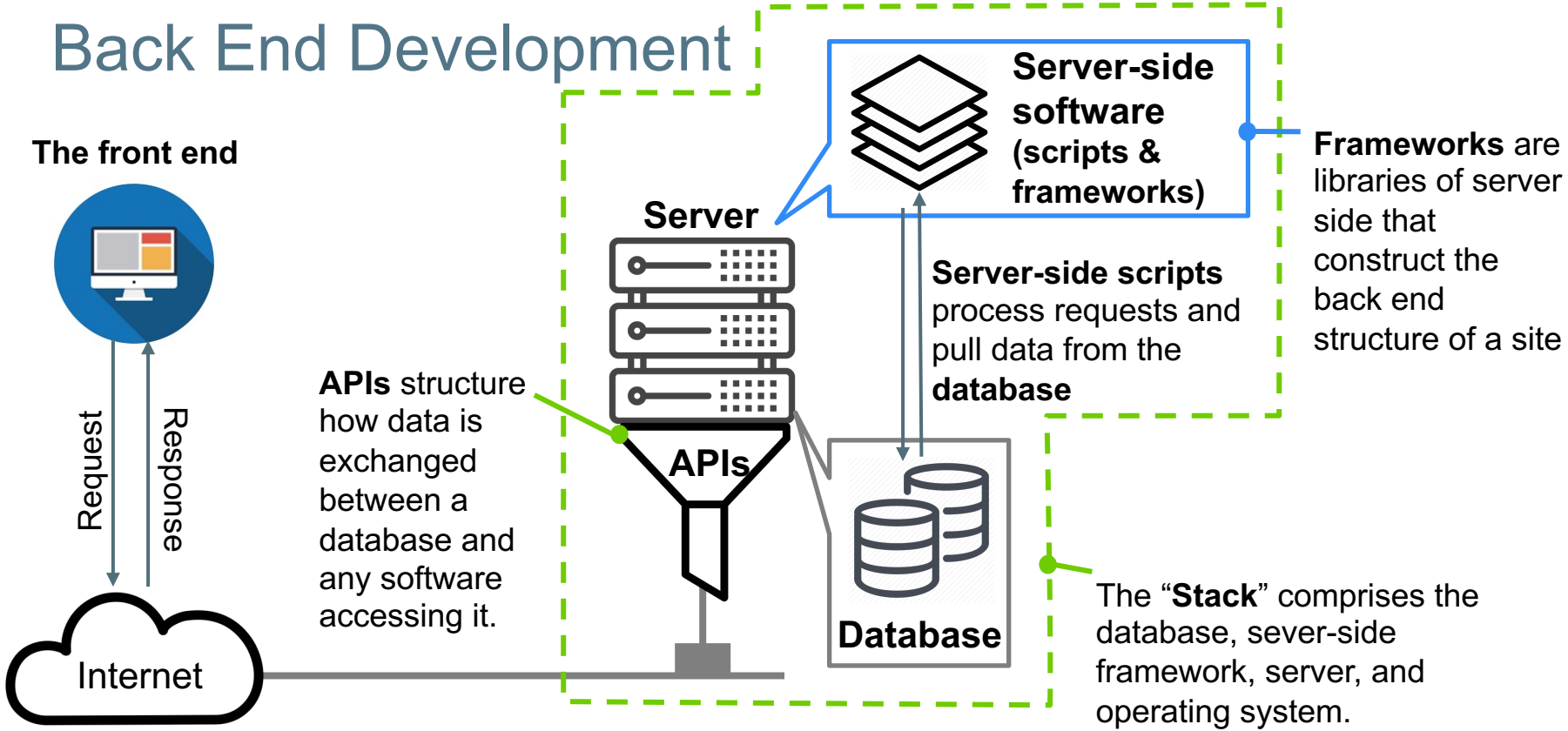
Q6&7: Your understanding of back-end frameworks

Back End Development

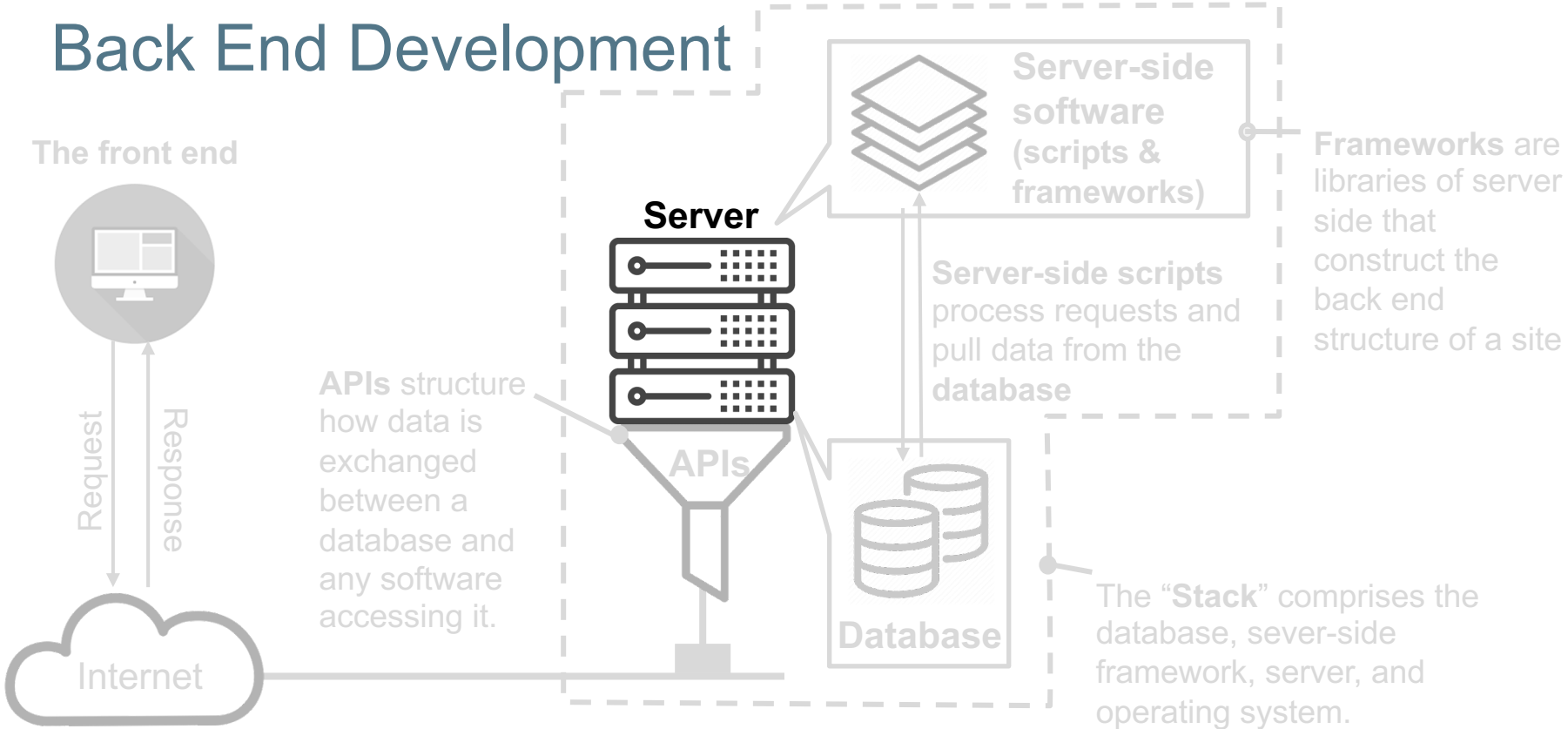
- Also called “Server-side” programming
- It is everything that happens on the server and databases
- Everything is behind the scenes



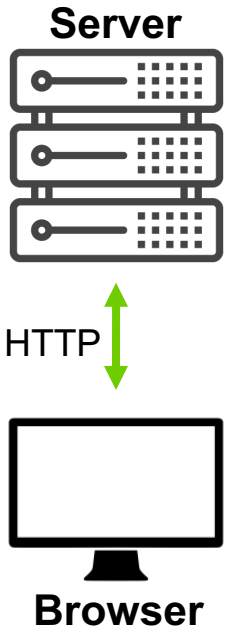
Back End Development



Back End Development



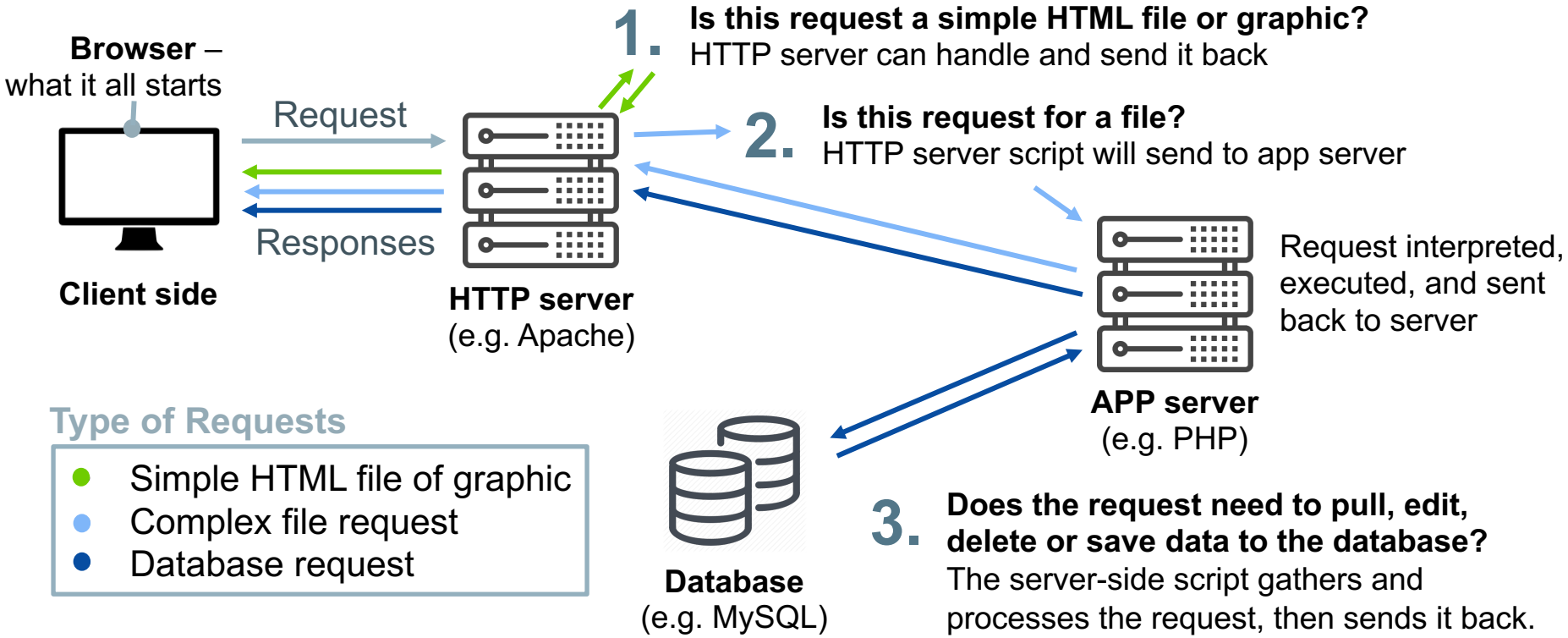
Server



What do servers do?

Browsers: send HTTP requests and get HTTP responses
Servers: get HTTP requests and send HTTP responses

Server: Behind the Scenes



Other servers

Mail servers

Sending and storing emails network.
e.g. Microsoft Exchange Server

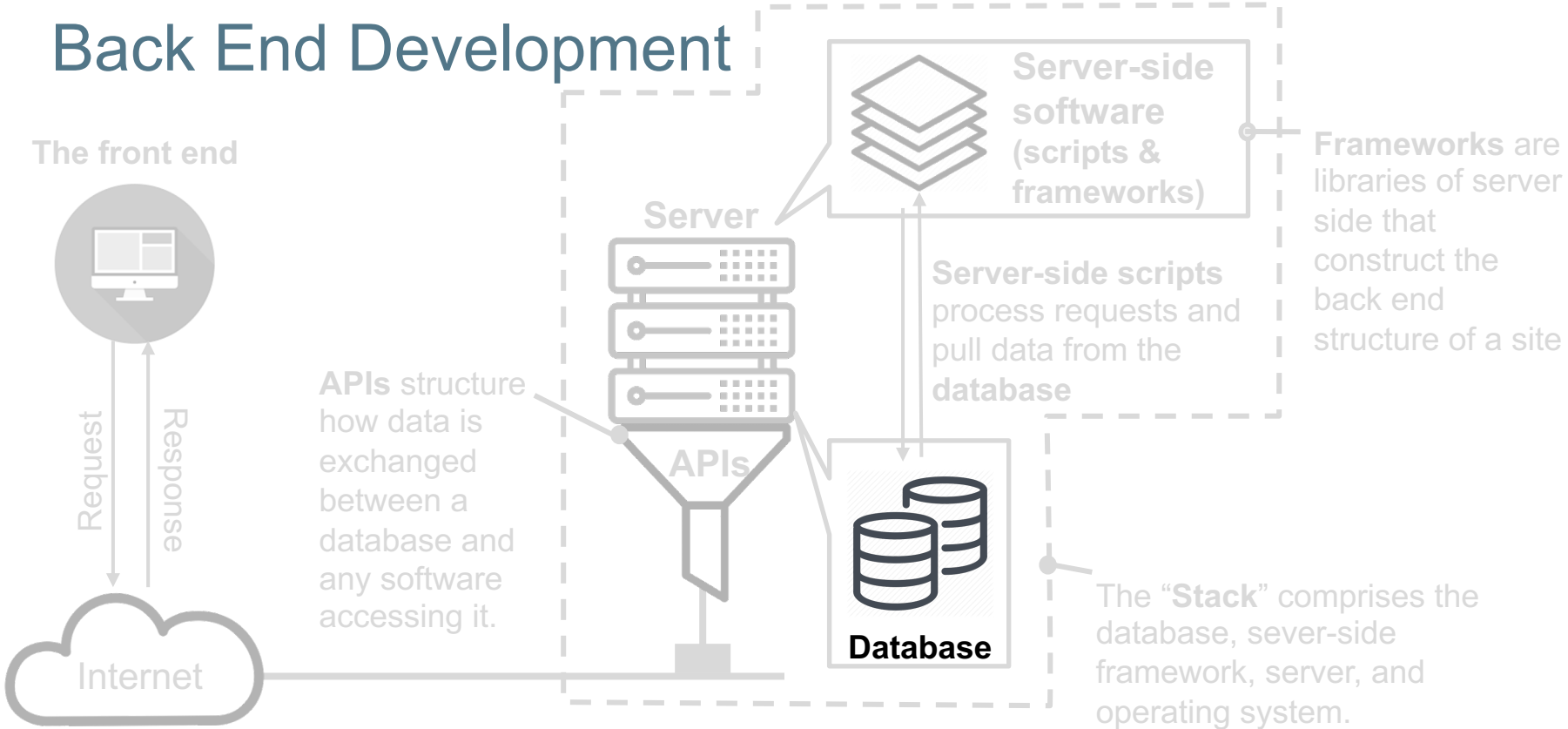
Proxy servers

These servers improve speed, security and performance between local network and the web by filtering requests and also providing cached versions of site pages to reduce network workload.
e.g. Nginx

FTP servers

All about uploading and sending files.
e.g. Filezilla FTPS

Back End Development



Database / Storage

Database

Databases store, organise and process information so that we can easily find what we need.

Properties

- Always available – Fetch correct app data and store updates
  - Even if many request come in concurrently – Scalable
  - Even if pieces fail – Reliable / fault tolerant
- Provide a good organisation of storing data
  - Quickly generate data for view
  - Handle app evolving over time

Database Type 1: Relational Database

Database

Relational Database

- Data is organised as a series of **tables** (also called **relations**)
- A table is made of **rows** (all called **records**)
- A row is made of a fixed (per table) set of typed **columns**

Relational Database Management System (RDBMS):

- **CRUD** functions: **C**reate, **R**ead, **U**pdate and **D**elete data
- Build-in programming language: **SQL** (Structured Query Language)
- e.g. MySQL, PostgreSQL, Microsoft SQL Server, Oracle

Ideal for

Organising and retrieving structured data

Database Type 2: NoSQL Database

Database

NoSQL Database

- NoSQL = “Not only SQL.”
- These databases are non-relational and distributed, addressing the issue that most modern data from the web is not structured.

How do NoSQL databases work?

Document-oriented: If a blog used a NoSQL database, each file could store data for a blog post: social likes, photos, text, links etc

Pros: Flexible & Ease of access (execute queries without SQL)
Cons: Require extra processing effort and more storage

Ideal for

Organising and retrieving inconsistent/incomplete data

SQL or NoSQL

Database

Reasons to use a SQL database

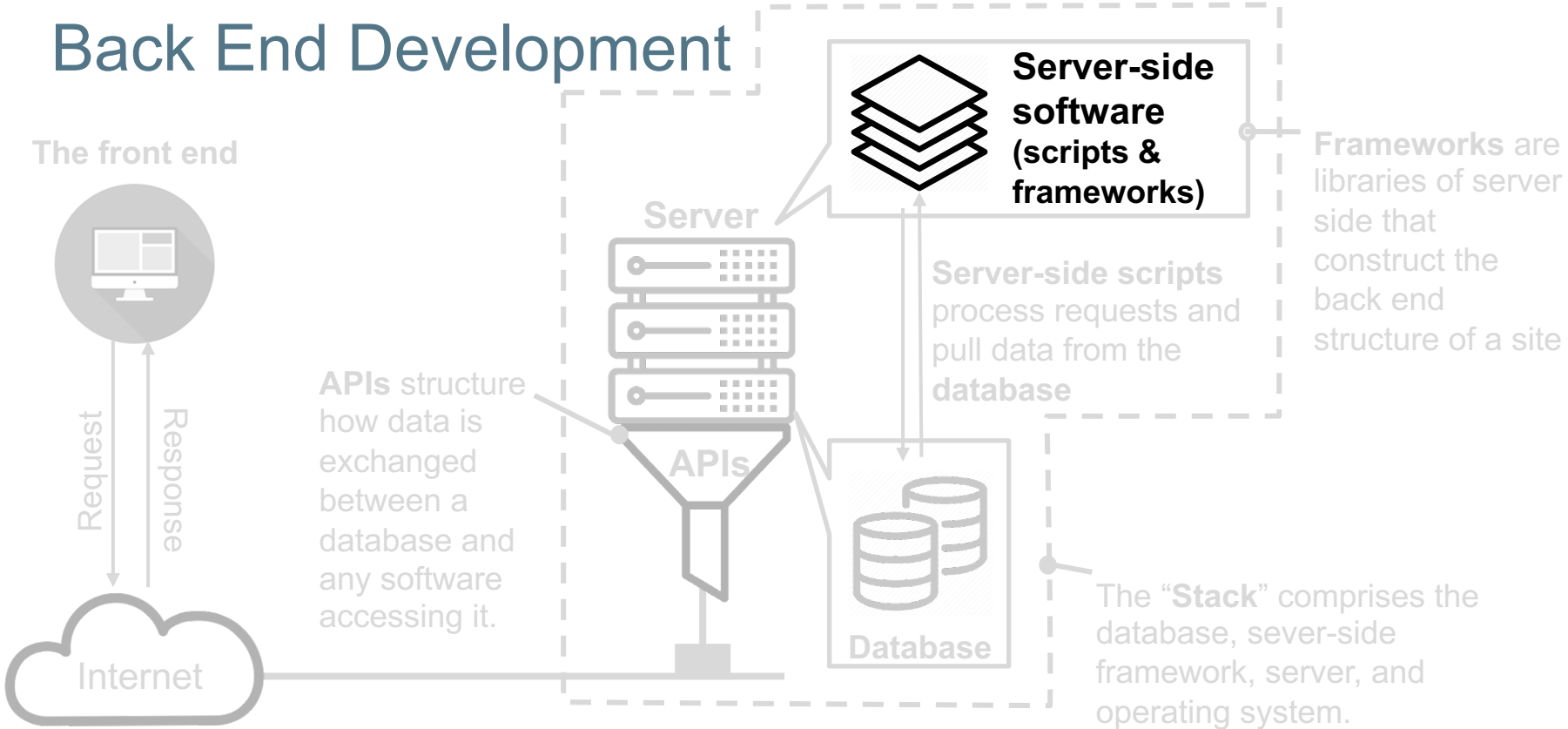
Your data is structured and unchanging.

Reasons to use a NoSQL database

1. Storing large volumes of data that have little or no structure.
2. Making most of cloud computing and storage
3. Rapid development

Source: SQL vs NoSQL

Back End Development

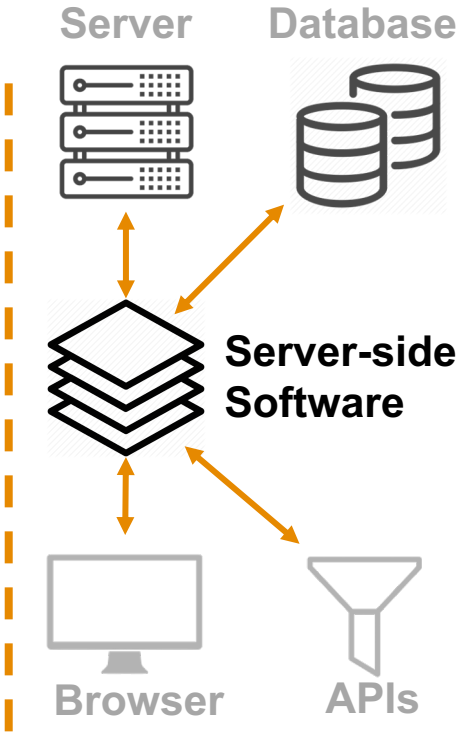


Server Side Software

- Also called “back-end web application”

Tasks

1. Provide application services based on business logic
2. Create the communication channel between browsers, server and storage system.
3. Build application programming interfaces (APIs), which control what data and software a site can share with other apps

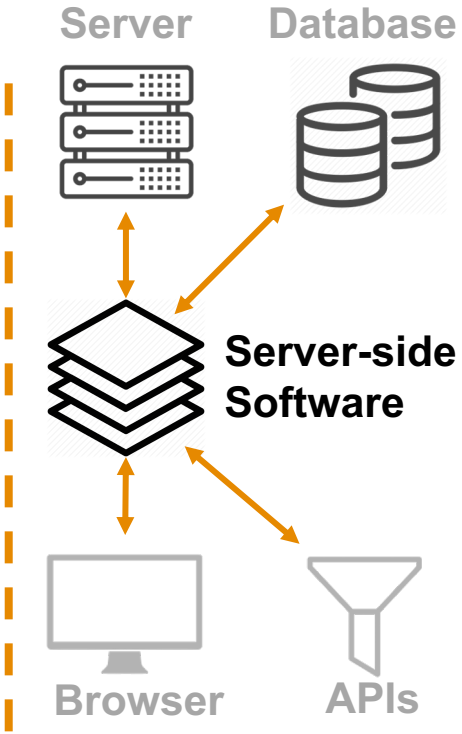


Server Side Software

- Also called “back-end web application”

Tasks

1. Provide application services bases on business logic
2. Create the communication channel between browsers, server and storage system.
3. Build application programming interfaces (APIs), which control what data and software a site can share with other apps

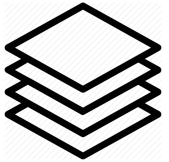


Server Side Software - Frameworks

A framework is a standardized set of concepts, practices and criteria for dealing with a common type of problem.

Why frameworks?
They boost performance, extend capabilities, and offer coding shortcuts that developers don't need to start from scratch.

Example:
When you're making a sandwich, it's much easier to buy pre-made, sliced bread from the store than it is to bake it on your own from scratch. Frameworks are your site's sliced bread—they speed up the process.



Server-side Software

Server Side Software - Frameworks

Generation 0: Before frameworks (1993-)

Browser: static HTML files only with HTML forms for input
Server: Common Gateway Interface (CGI)

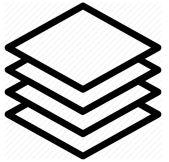
- Certain URLs map to executable programs to process data or generate pages
- Language: perl, c, c++, java

Example

Browser: `<FORM METHOD=POST ACTION=http://www.example.com/cgi-bin/formprocess.pl>`

Server: Execute "foromprocess.pl" from cgi-bin/

.pl: a program written in Perl (.java for java, .c for c)



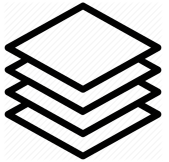
Server-side Software

Server Side Software - Frameworks

Generation 1 (1995-)

Server:

- Examples: PHP, Asp.net, Java servlets
- Incorporate language runtime system directly into web server
- Web-specific library packages were available:
  - URL handling, HTML generation, database access…
- Introduce **Templates**:
  - HTML/CSS describes view



Server-side Software

Server Side Software - Frameworks

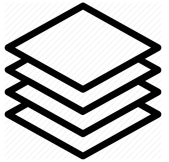
Generation 2 (2002-)

Server:

- Examples: Django for Python, Rails for Ruby, Laravel for PHP

- **Model-View-Controller (MVC)**
  - Divide an application into 3 interconnected parts

- **Object-Relational Mapping (ORM)**
  - Make database tables and rows appear as classes and objects



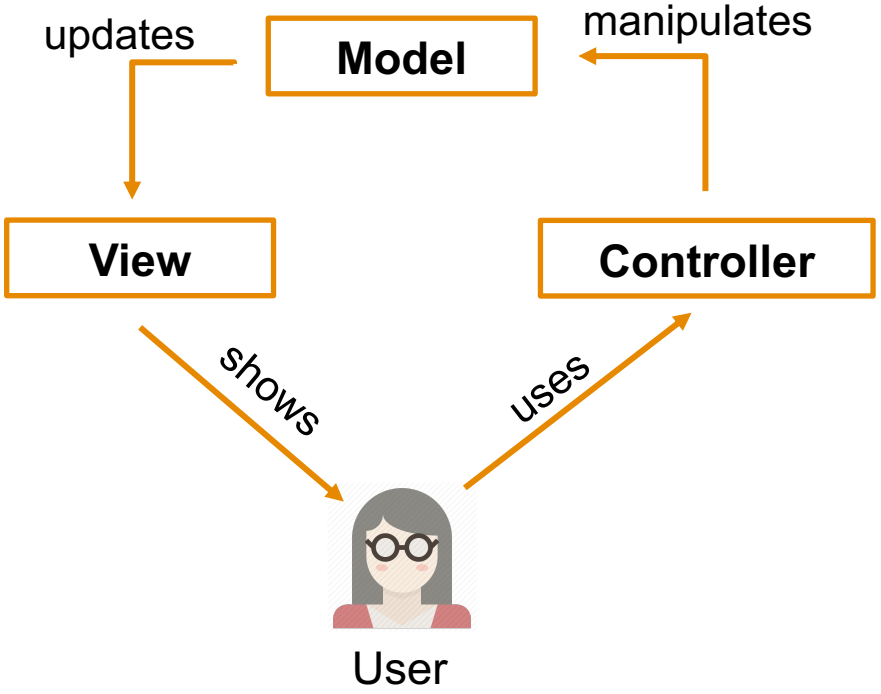
Server-side Software

Model-View-Controller (MVC) Pattern

Model:
- Manages application's data
- Connects the View and the Controller

View:
- Displays the web pages

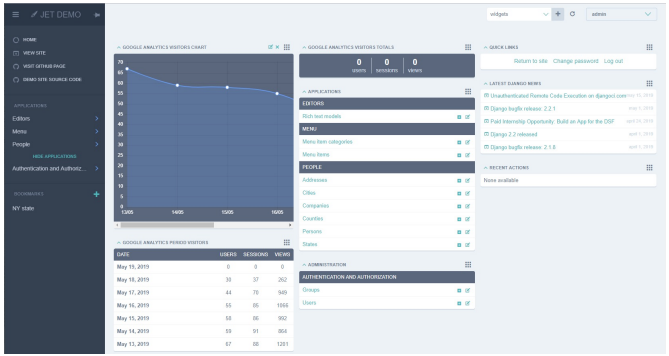
Controller:
- Handles users' interactions
- Fetches models and controls views

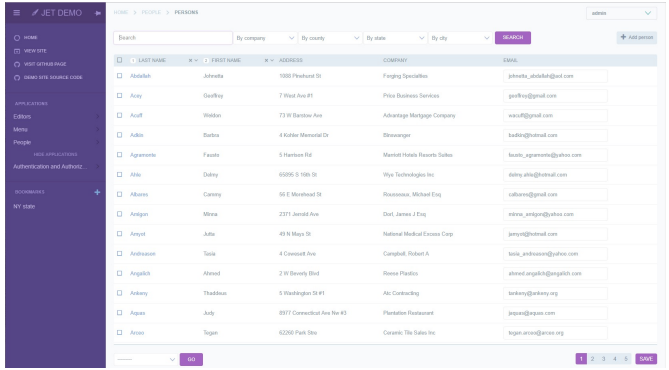


Model-View-Controller (MVC) Pattern

View Generation

- **Templates** are commonly used
  - Write HTML containing parts of the page that are always the same
  - Add bits of code that generate other parts that are computed for each page
  - The template is expanded by executing code snippets, substituting the results into the document
- **Pros**
  - Reusability
  - Easy to see how dynamic data fits in





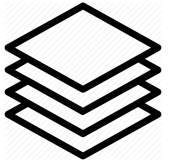
Server Side Software - Frameworks

Generation 3 (2010-)

- Examples: AngularJS

- JavaScript frameworks running in browser – More app-like web applications
  - Excels at building dynamic, single page web apps (SPAs)
  - Interactive, quick responding applications – Don't need server round-trip

- Many concepts of previous generations carry forward
  - Model-View-Controllers
  - Templates



Server-side Software

Server Side Software - Frameworks

Angular JS

- **Two-way data binding**

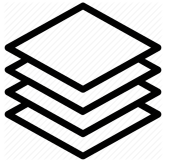
  A model variable is bound to a HTML element that can both change and display the value of the variable.

Source: Demo

The firstName and lastName model variables were bound to a couple of HTML input elements.

When the page is loaded, the value of the input elements are initialized to those model variables.

Whenever the user types something in an input, the value of the model variable is modified as well.



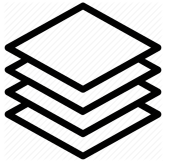
Server-side Software

Server Side Software - Frameworks

Generation 4 (2013-)

- Examples: React.js, Vue.js, Angular v2

- Focus on JavaScript components rather than pages/HTML
  - Views are reusable JS components rather than pages
  - Involve with more software engineering focus: modular design, reusable components, testability etc



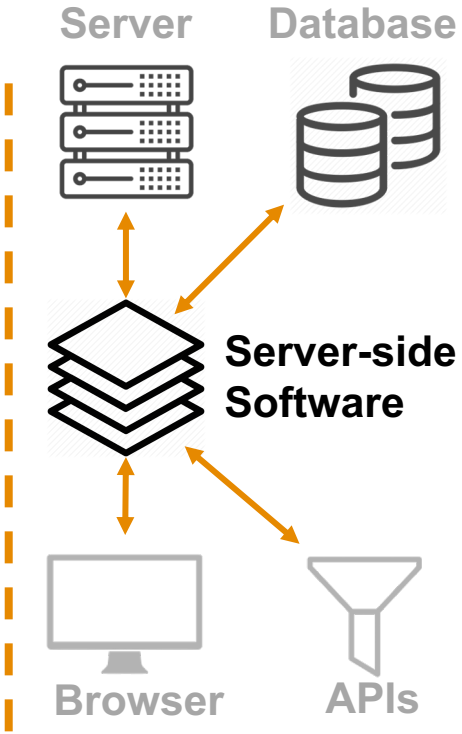
Server-side Software

Server Side Software

- Also called “back-end web application”

Tasks

1. Provide application services bases on business logic
2. Create the communication channel between browsers, server and storage system.
3. Build application programming interfaces (APIs), which control what data and software a site can share with other apps

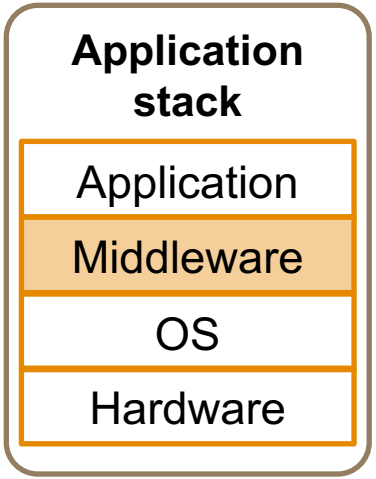


Server Side Software - Middleware

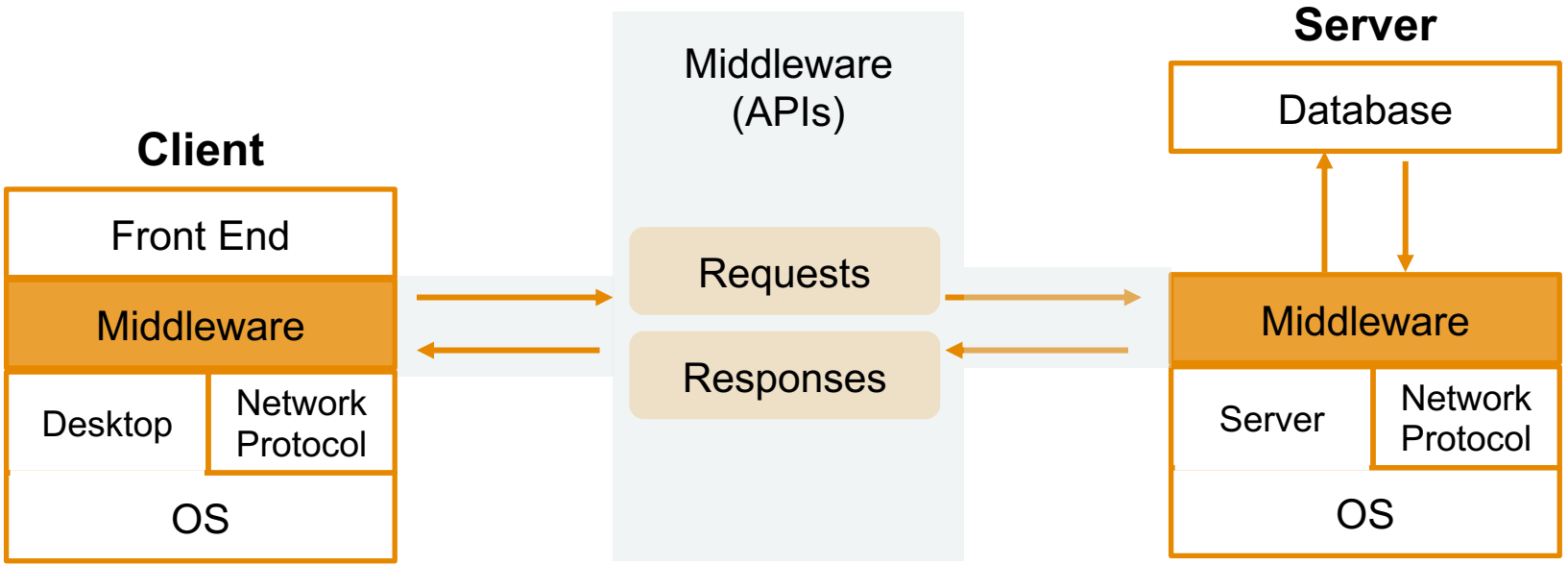
Middleware is any software that glues an application and its network.

Pros

- It facilitates client-server connectivity, forming a middle layer in the application that acts as “glue” between the app(s) and the network.
- It ties together complex systems and keeps all of the business’ software linked and able to communicate smoothly.
- It lets cloud applications and on-premise applications “talk” and provides services like data integration and error handling.



Server Side Software - Middleware

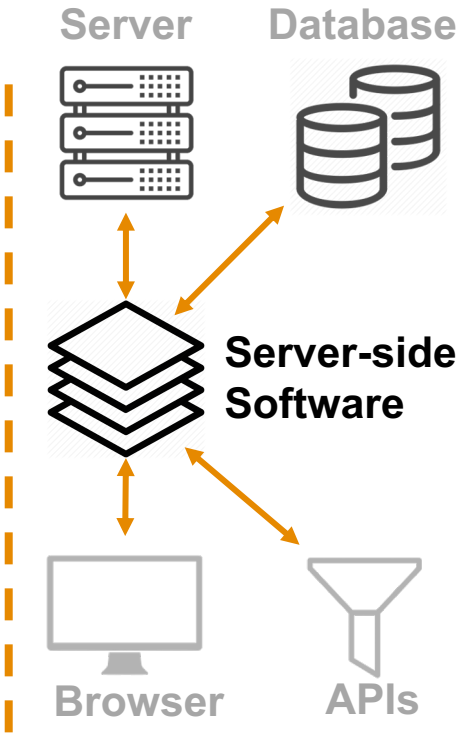


Server Side Software

- Also called “back-end web application”

Tasks

1. Provide application services bases on business logic
2. Create the communication channel between browsers, server and storage system.
3. Build application programming interfaces (APIs), which control what data and software a site can share with other apps



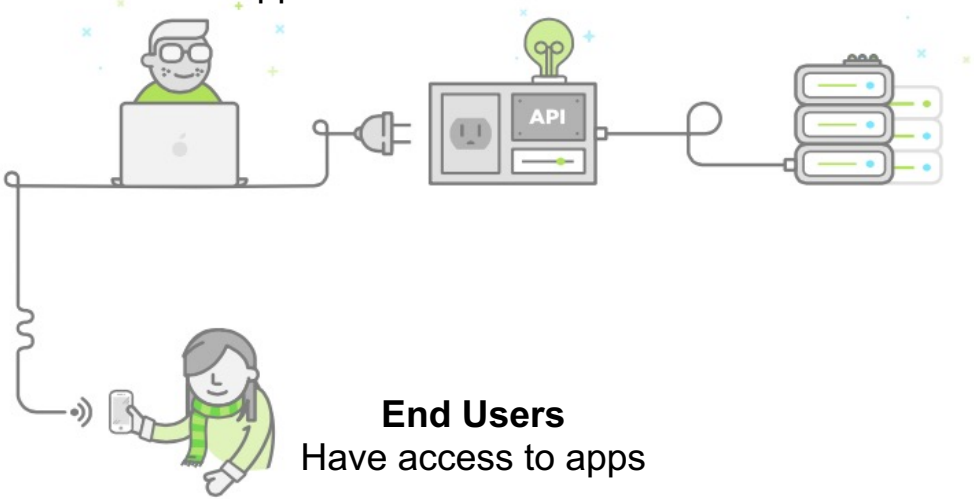
Server Side Software - APIs

An interface that allows two applications to talk to each other.

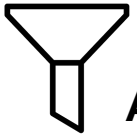
Developers
will access assets through API to build Mobiles or Web Apps

API
provides universal access to sharable assets

Assets
Data and software leveraged by developers and 3rd services

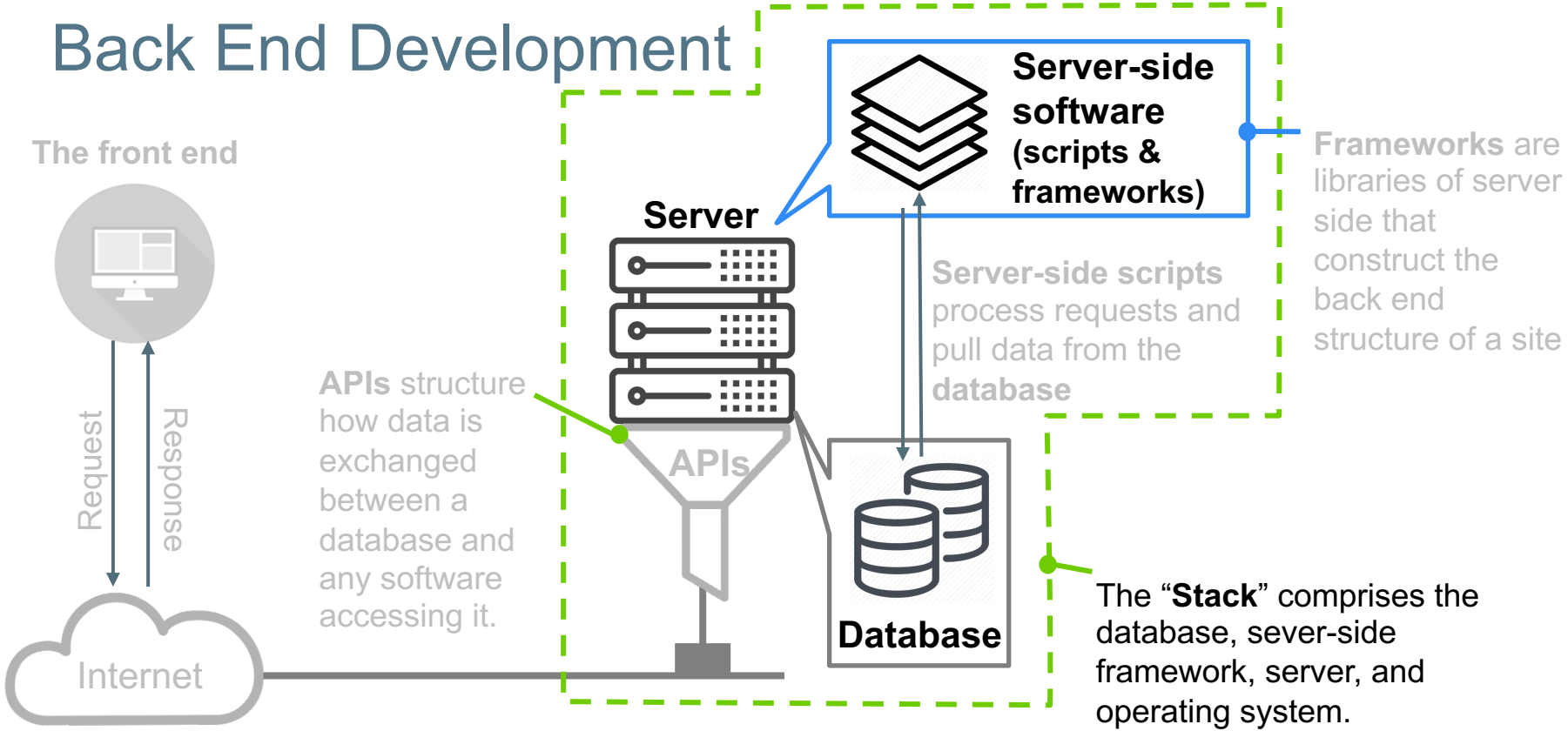


End Users
Have access to apps



APIs

Back End Development



Poll Everywhere

https://pollev.com/xuanyingzhu488

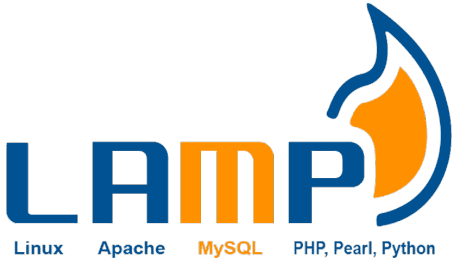
Q8: Your understanding of stacks

Common Stacks - LAMP

L: Linux operating system
A: Apache web server
M: MySQL database
P: PHP/Python/Perl application software

Pros

- Flexible, customizable, easy to develop and deploy
- A huge support community since it's open-source.
- Great for organizing massive amounts of structured data.



Variations

- WAMP: Windows/Apache/MySQL/PHP
- MAMP: Mac OS X/Apache/MySQL/PHP
- LAPP: Linux/Apache/PostgreSQL/PHP

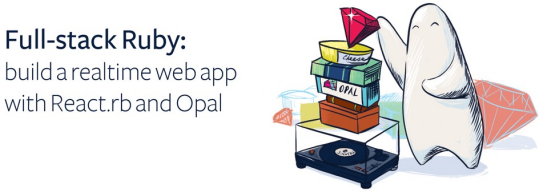
Common Stacks – MVC Stacks

Django Stack: Python / Django / Apache / MySQL

- Rapid development, Simplify deploying Django software

Ruby Stack: Ruby / Ruby on Rails / Apache / MySQL

- Rapid development



Common Stacks - MEAN

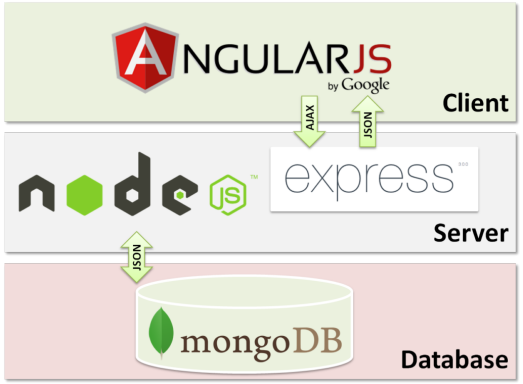
M: MongoDB
E: Express.js
A: AngularJS
N: Node.js

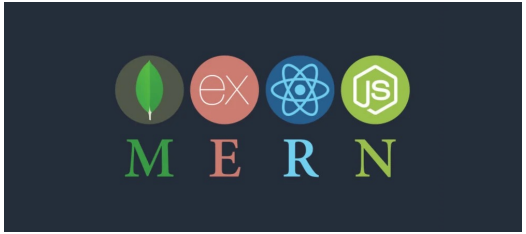
Pros

- Supports the MVC pattern
- Language uniformity.
- Document-based NoSQL database: more flexibility with semi-structured data.

Variation

- MERN: MongoDB / Express.js / React.js / Node.js

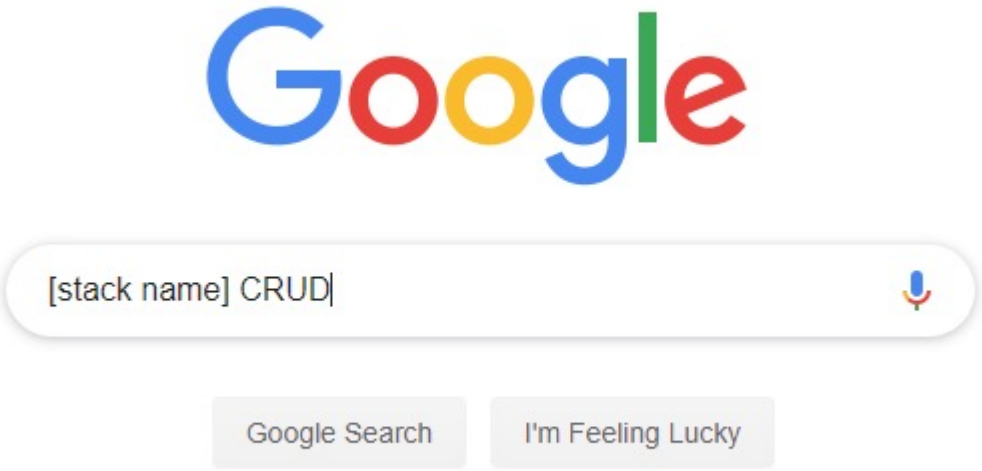




Source: MEAN

Tips: a way to learn stacks

Google: stack name + “CRUD”



Tips: a way to learn stacks

- LAMP: https://www.taniarascia.com/create-a-simple-database-app-connecting-to-mysql-with-php/

- MVC stacks:
  - Django: https://rayed.com/posts/2018/05/django-crud-create-retrieve-update-delete/
  - Laravel: https://itsolutionstuff.com/post/laravel-57-crud-create-read-update-delete-tutorial-example-example.html
  - Ruby on Rails: https://medium.com/@nancydo7/ruby-on-rails-crud-tutorial-899117710c7a

- MEAN: https://appdividend.com/2018/11/04/angular-7-crud-example-mean-stack-tutorial/
- MERN: https://codingthesmartway.com/the-mern-stack-tutorial-building-a-react-crud-application-from-start-to-finish-part-1/

Full Stack Web Application Architecture