Date: September 5, 2022

apsara

Date: __/__/____

WEEK 1

What Is An App?

What is an App?

→ An app is computer software, or a program, most commonly a small, specific one used for mobile devices.

Desktop Apps

Examples: MS Word
Web Browser
Visual Studio

(1) Usually standalone
(2) Often work offline
(3) Software Dev kits (SDK) → Custom frameworks; OS specific
↳ SDKs integrate with other parts of the machine such that the user doesn't have to carry out any integration manually.

Mobile Apps

(1) Targeted at mobile platforms: phones / tablets
(2) Constraints:
- Limited Screen Space
- User Interaction (touch, audio, camera)
- Memory / Processing
- Power
(3) Frameworks
- OS specific
- Cross-platform
(4) Usually network oriented.

SDK : a basic SDK will include a compiler, debugger & application programming interfaces (APIs), & much more.

apsara

Date: __/__/__

Web Apps

(1) The Platform
(2) Works across OS, device : create a common base
(3) Heavily network dependent

Components of An App

Components of An App

(1) Storage
(2) Computation
(3) Presentation

Example : Email Client

1. STORAGE
   - → where are the emails stored ?
   - → how are they stored on the server ? file formats etc
2. COMPUTE
   - → Indexing of emails
   - → Searching ; Filtering out
3. PRESENTATION
   - → Display list of mails
   - → Rendering / display of individual mails

Platforms

- Desktop
- Mobile
- Web-Based
- Embedded

Desktop

- → Keyboard, Mouse, Video
- → Desktop Paradigm - folders, files, documents

Mobile

- → Touchscreen
- → Voice, tilt, camera interfaces
- → small self-contained apps

Web-Based

- → Datacenter storage - persistent
- → Cloud : access anywhere, multi-device

Embedded

An embedded application is only used to perform a SPECIFIC TASK!!

- → single function, limited scope

Date
Sept 6, 2022

Tuesday

Date:

Architectures

→ Client - Server
→ Distributed Peer-to-Peer

Client - Server Architecture

(1) SERVER
- → stores data
- → provides data on demand
- → may perform computations

(2) CLIENTS
- → End Users
- → Request Data
- → User Interaction, Display

» Network :
1. Connects Client to Server
2. Can be local
3. Data pipe - no alterations

CLIENT - SERVER MODEL

→ Explicit differentiation between clients & servers

→ Local Systems :
- Both client and server on same machine
- Conceptually still a networked system

Page No.:

→ Machine Clients
- Eg. software / antivirus updaters
- Need not have user interaction

→ Variants
- Multiple servers, single queue, multiple queues, load balancing frontends

Examples :
(1) Email
(2) Databases
(3) Whatsapp / messaging
(4) Web Browsing

Distributed (Peer-to-Peer) Model

- All peers are considered "equivalent"
  → but some peers may be more equal than others
- Error Tolerance
  → Masters / introducers
  → Election / re-election of masters on failure
- Shared Information

Examples :
(1) Bittorrent
(2) Block-chain based systems
(3) IPFS, Tahoe (distributed file systems)

MVC
MVA
MVP
HMVC
....

Software Architectures

What is a design pattern ?

→ A general, reusable solution to a commonly occurring problem within a given content in software design.

- Experienced designers observe "patterns" in code
- Reusing these patterns can make design and development faster
- Guide the design and thought process

M-V-C Paradigm

-> MODEL
- → Core data to be stored for the application.
- → Databases ; indexing for easy searching, manipulation

-> VIEW
- → User-facing side of application
- → Interfaces for finding info., manipulating

-> CONTROLLER
- → "Business Logic" - how to manipulated data

Model
Store emails on server, index, ready to manipulate

VIEW
Display list of emails; Read individual emails

CONTROLLER
Sort Emails ; Delete ; Archive

→ User uses CONTROLLER → to manipulate MODEL → that updates VIEW → that user sees

Other Design Patterns

- Model - View - Adapter (MVA)
- Model - View - Presenter (MVP)
- Model - View - Viewmodel (MVV)
- Hierarchical MVC
- Presentation - Abstraction - Control

Each has its uses, but fundamentals are very similar

Date: September 7, 2022

Wednesday

Date:

Introduction to Web

Why the 'Web' ? 'Cross platform OS'

→ Generic - works across operating systems, hardware architectures
→ Built on sound underlying principles
→ Worth understanding
  ↳ Constraints : what can & cannot be done
  ↳ Costs : storage, network, device sizing, datacenter

HISTORICAL BACKGROUND

- **Telephone Networks** ~ 1890+
  → circuit switched - allow A to talk to B, through complex switching network.
  → Physical wires tied up for duration of call even if nothing said. # no body else can use them if A & B using.

↳ direct set of wires making a closed loop for A & B

- **Packet Switched Networks** ~ 1960s
  → Wires occupied only when data to be sent - more efficient use
  → data instead of voice (image, document)

A single wire b/w two exchanges can probably carry several different conversations.

→ The idea of PSN is that you could now take the voice data, break it up into small packets of information after digitizing. You convert it into a set of numbers & arrange them into packets. And now those packets are sent across the wire instead of the direct analog signal corresponding to a voice.

- **ARPANet - 1969**
  → Node to Node Network
  → Mostly university driven

Advanced Research Project Agency Network

Page No.:

- Internet came into existence on Jan 1, 1983
- TCP/IP → backbone of Internet

apsara

Date: __/__/____

- Protocol
  - → How to format packets; place them on wire; headers / checksums;
  - → Each network had its own protocol

- 'Inter' Network
  - → How to communicate between different network protocols?
  - → Or replace them with a single "inter"net protocol

- IP : Internet Protocol (1983) → standardized protocol for diff. kind of networks
  - → Defines headers, packet types, interpretation
  - → can be carried over different underlying networks: ethernet, DECnet, PPP, SLIP

- TCP : Transmission Control Protocol (1983)
  - takes care of reliable transmission → Establish reliable communications - retry; error control
  - → Automatically scale and adjust to network limits → How fast can you transmit information?

- Domain Names ~ 1985 → set up hierarchical structure → Root servers → Sub Domain servers
  - → use names instead of IP addresses
  - → easy to remember - .com revolution still in the future

- HyperText ~ 1989+
  - → Text documents to be saved
  - → Formatting hints inside document to 'link' to other documents - HyperText → led to the creation of WWW

- Tim Berners Lee - The World Wide Web → documents that are connected to each other

network of servers → Internet

Page No.: _____

CGI : Common Gateway Interface → interfacing with the network servers

apsara

Date: __.__.____

WHERE ARE WE NOW ?

- Original Web limited :
  - → static pages
  - → complicated executable interfaces
  - → limited styling
  - → browser compatibility issues

- Web 2.0 ~ 2004+
  - → dynamic pages
  - → HTTP as a transport mechanism - binary data; serialized objects
  - → client side computation and rendering
  - → platform agnostic OS
    - → cross-platform (can work in any OS)

WEB SERVER

- Any old computer with a network connection can act as a web server

- SOFTWARE :
  - → Listen for incoming network connections on a fixed port
  - → Respond in specific ways
  - → Opening network connections, ports etc already known to OS

- PROTOCOL : (how to communicate b/w two devices)
  - → What should client ask server
  - → how should server respond to client

The standardized part which says how the client should talk to the server & how the server should respond, is what is called an HTTP.

⇒ ports are given 16 bit value

301 code : page has moved
404 code : page is missing

apsara

Date: __/__/____

HTTP

- Hyper Text
  - → Regular text document
  - → Contains "codes" inside that indicate special functions - how to "link" to other documents
- Hyper Text Transfer Protocol
  - → Largely text based : client sends requests, server responds with hypertext document

Simple Web Server

SIMPLEST SERVER

```
while true ; do
    echo -e "HTTP/1.1 200 OK\n\n $(date)" |
    nc -l localhost 1500 ;
done
```

- while true ; do → keep on repeating whatever is inside indefinitely, until we use some other mechanism to kill this program
- line 1 / line 2
- 200 → http status code
- | → this pipe symbol means that do something with line 1 & pipe that info into line 2.
- done → closing

- → This has been written in shell script.
- → echo -e : used to output on screen whatever is inside the quotes
- → | : the pipe symbol redirects the standard output to second line
- → HTTP/1.1 : this server is responding with an http 1.1 code.
- → 200 OK : it is the code number. The 'status' code
  - ↳ it means everything is OK
- → \n : leaves one line (that is how HTTP header & data are separated)
- → $(date) : gives out the 'date' info

Page No: _____

→ So, basically, this echo command in other words is going to print out the present date.

But in the above code, the pipe (|) symbol redirects it into the second command, which is nc.

→ nc → Netcat
Netcat is going to listen on localhost, ie, the local server, port number 1500

So, that is it !!, That is what the server is now doing.

→ nc -l localhost 1500 : implementing the server
↳ it means that nc will cause the OS to open a port and will listen and anytime a request comes in on port no. 1500, that will be fed to nc.

What does 'nc' do ?
→ It will just take whatever came from the echo & send it out to whoever asks anything on port 1500. It does not interpret the request.

GENERAL WEB SERVER

- listen on a fixed port
- On incoming request, run some code & return a result
  → Standard headers to be sent as part of result
  → Output can be text or other format - MIME

curl http://localhost:1500] command to connect to server

↳ it allows you to create custom requests to a web server. more functionality like sending right headers etc.

apsara

Date: __/__/____

TYPICAL REQUEST (from curl to the web server)

```
GET / HTTP/1.1          → actual request
Host : localhost:1500
User-Agent : curl/7.64.1   } optional
Accept : */*
          ↳ MIME type (accept anything)
```

curl SIMPLE WEB SERVER

```
curl -v http://localhost:1500
```

→ IP address → ipv6 (the version 6 adress)

```
1  *   Trying ::1...        → double colon means localhost means your own machine
2  * TCP_NODELAY set       → TCP tuning settings → controls how fast the connection will be set up
3  * Connected to localhost (::1) port 1500 (#0)   → connection established
4  > GET / HTTP/1.1        → it says that I want to get some information from you. slash indicates that it wants basic core info; the route info.
5  > Host : localhost:1500
6  > User-Agent : curl/7.64.1
7  > Accept : */*
8  >                       → blank line indicates that rqst is done (it is important)
9  < HTTP/1.1 200 OK       → everything is fine
10 * no chunk, no close, no size. Assume close to signal end
11 <
12   Wed Sep 7 08:44:55 IST 2022   → output/info we rqsted for
```

→ The lines starting with * are just debug information from curl, they are not the part of the protocol. It is not sent from the client to the server or back.

→ In line 3, if there was no server running on localhost, it would have said, either access denied or server did not respond.

→ line 10 → reached the end, did not get any further information.

→ line 4 → it is also mentioning that it is using HTTP/1.1 protocol

Date: September 8, 2022

Thursday

apsara

Date: ___.___.___

What is a Protocol ?

PROTOCOL

↳ how computers are expected to communicate with each other.

- Both sides agree on how to talk
  - → walk into a room, greet each other, shake hands, sit down, chat about weather

- Server expects "requests"
  - → Nature of request
  - → Nature of client
  - → Types of results client can deal with

- Client expects "responses"
  - → Ask server for something
  - → Convey what you can accept
  - → Read result & process

HYPER TEXT TRANSFER PROTOCOL : HTTP

- Primarily text based

- Requests specified as "GET", "POST", "PUT" etc.
  - → Headers can be used to convey acceptable response types, languages, encoding
  - → Which host to connect to (if multiple hosts on single server)

- Response headers
  - → convey message type, data
  - → cache information
  - → status codes : 404 not found

USE CASES

- GET : simple requests, search queries etc.
- POST : more complex form data, large text blocks, file uploads
- PUT / DELETE / ...
  - → rarely used in Web 1.0
  - → Extensively used in Web 2.0
  - → Basis of most APIs - REST, CRUD

ANOTHER WEB SERVER

python -m http.server

- Serves files from local folder
- Understands basic HTTP requests
- Gives more detailed headers & responses

→ It basically says take the module HTTP and run the function server from inside the HTTP module.
This is a one-line web server.

Date:

GET / (curl http://localhost:8000) GET/server.py

debugging info

```
*   Trying ::1 ...
* TCP_NODELAY set
* Connected to localhost (::1) port 8000 (#0)
> GET / HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.64.1
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Server: SimpleHTTP/0.6 Python/3.9.2
< Date: Thu, 8 Sept 2022 03:11:15 GMT
< Content-type: text/html
< Content-Length: 31
< Last-Modified: Thu, 8 Sept 2022 03:10:18 GMT
<
<h1> Hello World </h1>
Hi There!
* Closing Connection 0
```

what was sent by curl

it says it is a simple HTTP server with Python 3.9.2 version.

→ MIME type of result

it basically says treat whatever you are getting now as an HTML file, which means basically display it.

Date: September 9, 2022

apsara

Date: ___/___/___

Performance of Website

LATENCY

→ Latency refers to how much time does it take to get the response for a given request.

→ Speed of light : $3 \times 10^{+8}$ ms^{-1} in vacuum ; 2×10^{8} ms^{-1} on cable
- ~5ms/km ⇒ ~5ms for 1000km

Example: We have a data center 2000km away. One way, the request is going to take 10 milliseconds which means that the round trip will take 20 m.s.
So what does that mean?
It means that if I am continuously sending requests to the serves, and I want it such that every time I send a request, I want to get back the respond before I can then proceed, I am limiting to a maximum of 50 requests per second.

RESPONSE SIZE

→ Response = 1KB of text (headers, HTML, CSS, JS)

→ Network Connection = 100 Mbps (MegaBits per second)
- ~ 10 MBytes/sec

→ ~ 10,000 requests / second

EXAMPLE

Google Search Response

- Headers ~ 100B
- Content : 144KB
- Approx 60,000 requests per second (maybe)

~ 80 Gbps bandwidth

Memory - YOUTUBE

- One python HTTP server process : ~ 6MB (measured)
- Multiple parallel requests : multiple processes
  - → eg YouTube will have long running server processes
- 2016 Presidential Debate in US :
  - → >2 Million concurrent viewers

2M x 6MB ⇒ 12TB RAM

Storage - GOOGLE

- Index 100s of billions of web pages
- cross-reference, pagerank
- Total index size estimate :
  - → 100,000,000 Gigabytes ⇒ 100 Petabytes
- storage ?