# Lecture-9: Telemetry

# What is telemetry

- Goal is “observability” (be able to observe user actions, automatically)
- Tele = distance / far / remote (as in telephone); metry = measure
- Telemetry is simply making measurements about users’ usage of an app/service/device/tool—and the data is sent to a “far-off” location to be analysed to generate insights.
- Common example:
  - Rainfall / air quality / air pressure: sensors / gauge measure in one place and transmit  the data to a meteorological station geographically away
- Two possibilities:
  - Measure and send raw data to server (e.g., every click and its metadata)
  - Send aggregates (e.g., count no. of clicks and send at some frequency)

# “Components” of telemetry

- App-side data collection
  - E.g., click stream, server traffic on a web server
  - Consider whether to send data / aggregate statistics
  - Plan for what questions & what data ahead of time

- Data transmission to Telemetry server
  - Via internet, typically
  - Consider frequency
  - Consider offline vs. online use
  - Plan for data transmission when you go out of network (store locally and send when back up. Typically managed through timestamps!)

# “Components” of telemetry (contd.)

- Data storage (on telemetry server side)
  - Consider a format that is convenient for analysis
  - Often needs a combination of data stores (key-value pairs, tables)
  - Consider storing timestamps and session details (Session ID, userID)

- Analysis
  - Simple aggregate statistics (e.g., sum, average, hourly, geographic, etc.)
  - A-B test (e.g., compare product views in two versions of app )
  - Recreate user episodes by stringing together events for a userID/session, by timestamp => for example, helps understand user behavior patterns
    - Fun fact: you can guess what a person might be looking for  on the web, from looking at the content in immediate browsing history (without ever looking at search query!)
  - Bin users into categories based on behaviours

# Examples of telemetry

- Google analytics
- Open telemetry
  - A standard, implemented by various libraries / languages / SDKs
- A ton of Javascript libraries
  - Analysis is something you have to do on your own
- Sometimes, people build their own

# How telemetry works

- Clicks, opens, etc. are called “Events” in programming
- There is code to collect “events”
- Typically, when an event happens, there is an event handler to mention behavior (e.g., open page “A” when “Button A” is clicked)
  - This is coded by programmers
- Telemetry works by:
  - Editing code for event handlers (“stitch in code for telemetry!”)
    - E.g., Change “open page A” to “save <time, userID, buttonA>; then open page A”
  - Sometimes this is by actually rewriting code, other times by generic settings
    - Listen to any event, save the <time, userid, event>, and then do what handler says
    - In turn, handler behavior also logged as event (e.g., <time, userid, “page opened event” “URL opened”>,

# Things to remember / consider

- Research questions / goals to be planned apriori
- Bake in robustness in telemetry
  - Consider cases of internet down, offline working, app crashes, etc. as you would for other aspects of app
- (For developers / engineers) Emphasize on clean code
  - Do not litter telemetry code everywhere, use libraries or build one if needed. Can lead to bugs, and slows down!
- Extensibility
  - Shouldn’t be too hard to add one more detail / parameter / element or event!

# Telemetry vs. logging

- Logging is also about collect data automatically, based on what is happening; overall goal is “monitoring” (monitor services)
- Typically,
  - In telemetry => data is gathered at one site, sent to a separate remote server for storage and analytics
    - E.g., google analytics → gathers data from your server / client browser, and sends to google servers (or separate configurable servers)
    - Usually, rich user-interaction data!
  - In logging => data is stored locally (same site as is collected!)
    - E.g., “attach logs” on windows crash, command line logs when something happens, security logging, server logs (requests and response status)
    - Smaller, less rich set of user-interaction data, usually!

# Uses of logging

- Debugging an issue, error and crash reports
  - Reporting where the code was, and state of some variables
- Detecting anomalies (e.g., security violations)
- Understanding user behaviours
  - Frequently accessed pages / services / times => can be used to make them easy to access, or provision better, or optimize performance
  - Look for “unfound” search result queries, and provide that information / add FAQs / optimize vocabulary
  - Look for user device / geography => show appropriate content / language
- Web servers provide for free; plenty of logging libraries
- Logs can be in flat files / databases / anywhere in between
- Think of what to log, and update as you go!