APIs

A mini-lecture series

CSE498 Collaborative Design - Secure and Efficient C++ Software Development

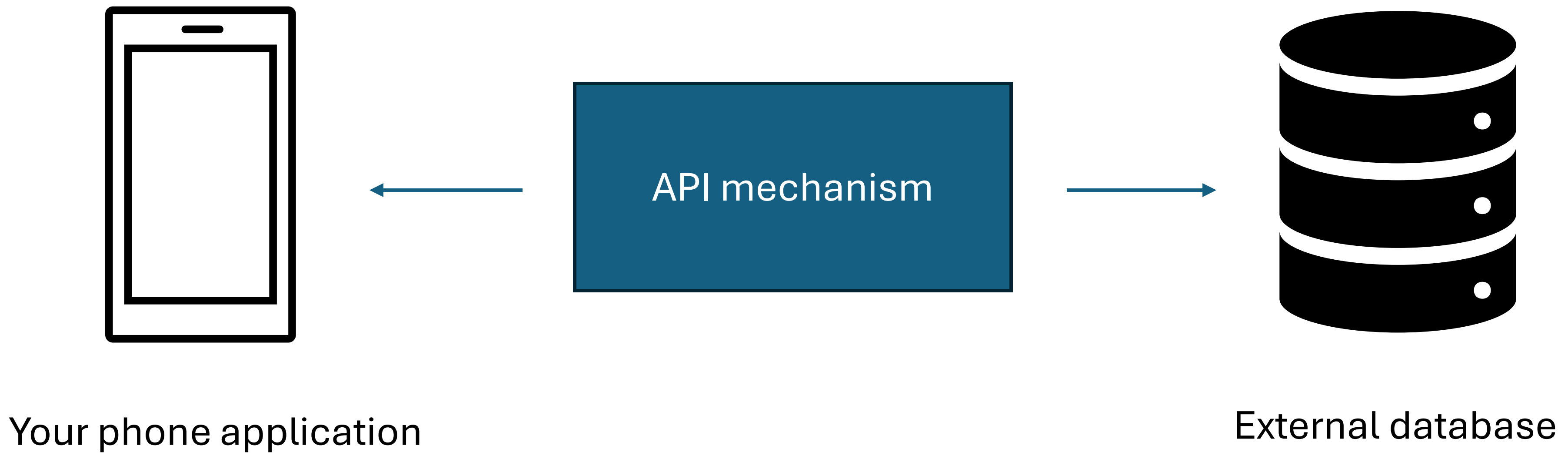
01/20/2025

Kira Chan

https://cse.msu.edu/~chanken1/

Application Programming Interface (API)

- Set of rules and protocols that allows two software components to communicate with each other



Example uses

- Online sales platform
- Weather applications that use data from public weather stations
- System components of a vehicle
- C++ libraries

- Really any two software that communicates between each other
    - API describes the protocols and format in which data has to be sent

Client-server model

- The application is the client, wishing to access some feature offered by the server
- The program that offers the service (the client), *exposes* certain endpoints to the “world”
- Client applications interact with the server via those endpoints

Key idea

- From server’s view: abstraction of complex functions and logic
  - Server has some data or functions
  - Client need only to supply input and get their outputs
  - The logic and algorithms used is irrelevant (or proprietary)

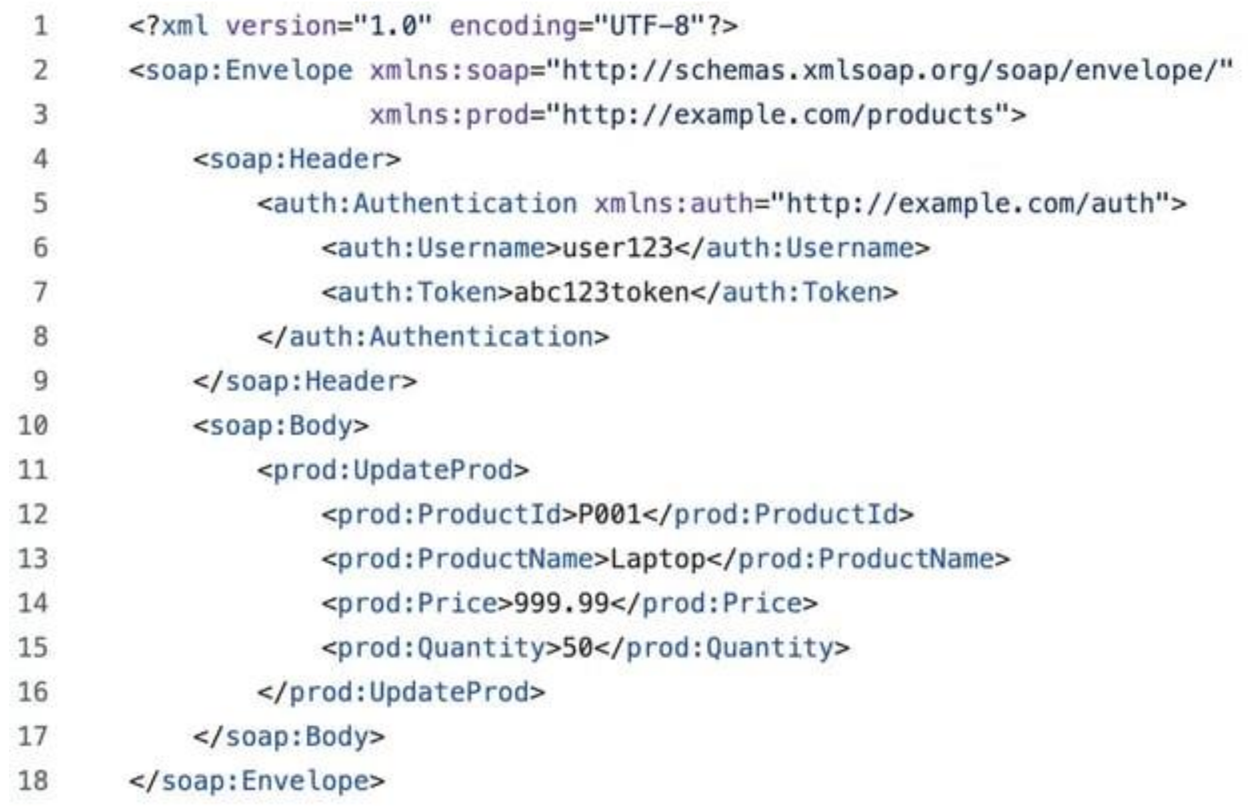
- Example: Coffee shop
  - You order a latte (barista is the API)
  - Magic happens behind counter
  - You receive your drink

Types of commonly used APIs

- Simple Object Access Protocol (SOAP)
- Remote procedure call (RPC)
- WebSockets
- Representational State Transfer (REST)

Simple Object Access Protocol (SOAP)

- Very simple protocol where messages are exchanged via XML



```xml
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:prod="http://example.com/products">
    <soap:Header>
        <auth:Authentication xmlns:auth="http://example.com/auth">
            <auth:Username>user123</auth:Username>
            <auth:Token>abc123token</auth:Token>
        </auth:Authentication>
    </soap:Header>
    <soap:Body>
        <prod:UpdateProd>
            <prod:ProductId>P001</prod:ProductId>
            <prod:ProductName>Laptop</prod:ProductName>
            <prod:Price>999.99</prod:Price>
            <prod:Quantity>50</prod:Quantity>
        </prod:UpdateProd>
    </soap:Body>
</soap:Envelope>
```

Source

Remote Procedural Call (RPC)

- Client completes a function on a server, and server returns the output
    - Client request the execution of a specific function with supplied parameters
    - Server acknowledges with response, executes function, client is blocked (waiting)
    - Server completes logic execution, returns result to client
- Exposes full function to the client (trusted)

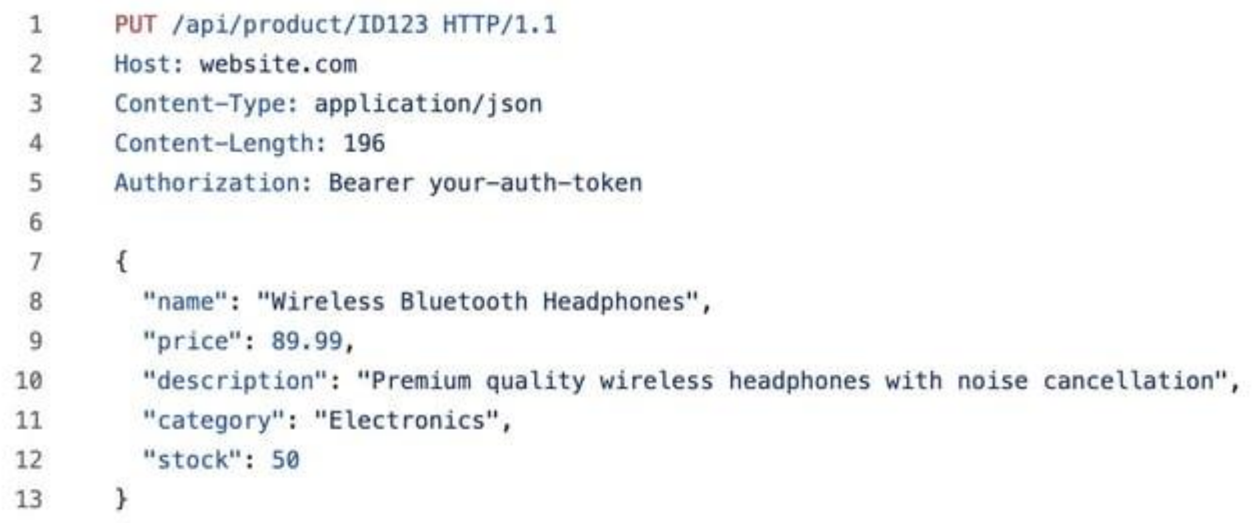
WebSocket APIs

- Establish a connection once between client and server (TCP)
- Keep the connection open and send messages as needed
  - Client OR server can send messages whenever they like
- Stateful: Keeps track of what happened previously
- Applicable to real-time applications

Representational State Transfer (REST)

- Very flexible way to build an API over HTTPs
- Stateless: each call is independent (a new request)
- CRUD:
  - HTTP calls (GET, POST, PUT, …) -> create, read, update, delete
- Can use any messaging format – typically JSON

Example of REST



```
PUT /api/product/ID123 HTTP/1.1
Host: website.com
Content-Type: application/json
Content-Length: 196
Authorization: Bearer your-auth-token

{
  "name": "Wireless Bluetooth Headphones",
  "price": 89.99,
  "description": "Premium quality wireless headphones with noise cancellation",
  "category": "Electronics",
  "stock": 50
}
```

Source

Challenges for building an API

- User friendliness
  - Does your API's function and parameters match user expectations
- How much do you want to expose to users?
- What format should you use?
- Design and clearly document the API (how it should be used)
  - Similar to documenting functions

- Flexibility vs conformity (size of userbase)

Challenges on using APIs

- Many companies may charge for API calls
  - Can range anywhere from few cents to even dollars per call
  - LLM models charge based on number of tokens processed as well
- You should minimize the number of calls if needed
- Avoid retrieving redundant or irrelevant data
- Create *mock servers* to test functions that involve API calls

For your projects

- Since each team will be responsible for developing a piece of the simulation world, you will need some mechanism to communicate
- E.g.,
  - Team A is responsible for building the world and generating content
  - Team B is responsible for building the agents
  - Team C is responsible for building the interface module (web interface)
- Example: Agent calls move right from the world, world returns updated cell location

For your projects

- Work with other teams within each company to conform to a consistent API format between each layer
- Because these involve C++ classes, you likely will not need other formats like SOAP, REST, etc.
- But you do have to agree on how other teams will use your function, and be ready to adapt to suit the needs of your customers (other teams)

Persons of the day – Maurice Wilkes and David Wheeler

- Worked on a modular software library for the EDSAC computer
- Created the first API documentations for how to use their library of software
    - Punch cards in filing cabinet
    - Library catalogue for notes on programs and instructions on how to use them