Interactive Digital Systems

Efterår 2022

Link til kode: https://editor.p5js.org/Laura321/sketches/PYuABuctV

Anna Borg	71703
Caroline Hornesby Vargas	72004
Christine Van Wulffeld	71998
Laura Sofie Juel Nielsen	72005

Indholdsfortegnelse

1. Koncept

En misset bus, tog eller blot lang ventetid når du skal med offentlig transport? Hvorfor ikke udnytte denne ventetid, til at se nyttige informationer om din rejse. Vores koncept er at vide rejseinformationer i realtid på en sjov og interaktiv måde.

Som bruger af offentlig transport er man højst sandsynlig bekendt med reklamesøjler, som forekommer på togstationer og busstoppesteder. Disse reklamesøjler kan alternativt anvendes til at skabe et interaktivt rum, i form af en skærm, som den rejsende kan interagere med. Idéen er at programmet, som kører på skærmen, viser informationer om vejr og dato, samt tog- og busafgange fra København H. Informationen bliver tilgængelig for den rejsende, ved at interagere med skærmen. Dette er implementeret således, at brugeren ved hjælp afposering, får vist den valgte information. Ved at "trykke" på vejrudsigt vil der blive vist et billede af en sol eller sky, svarende til det vejr der sendes fra den implementeret API. Desuden vil dette ikon ændre sig som temperaturen stiger eller falder. Ved at "trykke" på rejsetider vil der komme information om fremtidige tog afgange fra Københavns Hovedbanegård. Vores program henter informationer fra API'er på nettet og programmet anvender machine learning.

1.1 Valg af informationer der vises:

I den interaktive oplevelse er der taget udgangspunkt i at vise rejseinformation samt vejrudsigten.

2. Overordnet beskrivelse af programmet

Rapporten er struktureret således at der først forekommer en redegørelse, hvor vi vil gennemgå de begreber, der anvendes i programmet og som vi senere i rapporten gør brug af. Efterfølgende beskrives udvalgte funktioner, hvor disse uddybes. Slutligt vil der i konklussionen fremlægges mulige tanker om videreudvikling af konceptet.

Programmet er udviklet i P5JS, hvortil vi har implementeret machine learning "M15 PoseNet" fra Mads Hobyes hjemmeside (Hobye, n.d.). Programmet indeholder en række funktioner hvoraf setup() og draw() er P5JS hovedfunktioner. Derudover er der implementeret PoseNet funktioner og egne udarbejdede funktioner. Vi har foretaget ændringer i koden hentet fra Mads Hobyes hjemmeside. Disse ændringer er foretaget ud fra elementer, som vi har fundet relevante at vores pose tracker skal identificere.

2.1 Redegørelse

2.1.0 API

Da vi har valgt, at der på vores interaktive reklamesøjle skal fremgå informationer om vejret og rejsetider, har vi til dette anvendt netværks programmering ved at implementere to API'er (Application Programming Interface). En API tillader et stykke software, at kommunikere med et andet stykke software (REST API Architectural Constraints, u.d.). Programmet anvender to forskellige API'er: "openweather map" og rejseplanens API.

API'en "openweather map": Vi har valgt at programmet i første iteration henter data om vejret for byen København. Vi ønsker at programmets vejrinformationer, afhænger af byen, som programmet anvendes i.

Rejseplanens API: Denne API henter informationer fra Rejseplanen.dk og er specificeret til at hente 20 togafgange fra Københavns Hovedbanegård for d. 24/12/2022. API'en vi anvender henter "*DepartureBoard*", som bl.a henter informationer om Tidspunkt for afgang, slut destination, type af tog mm.

2.1.1 Files (Json, xml)

I programmet benyttes to forskellige filer når vi henter vores URL. Json (JavaScript Object Notation) (JSON, u.d.) som er baseret på JavaScript programming og XML (Extensible markup language), som overordnet er mere sikkert end Json (Difference between JSON and XML, u.d.). Vi har valgt ikke at gå yderligere ind i forskellen samt uddybelsen af de to filtyper, men er opmærksomme på at de er benyttet i vores program.

Openweather map's filtype: json

Rejseplanens filtype: XML

2.1.1 URL

Uniform Resource Locator (URL) er beskrivelsen af adressen på en bestemt ressource på internettet (*What Is a URL? - Learn Web Development*, n.d.). I vores program har vi lavet to globale variabler, som hver har værdien af den faktiske URL-adresse for API'erne. URL variablerne anvendes i en preload funktion, som henter dataen fra URL'en.

URL til "openweather map" API:
https://api.openweathermap.org/data/2.5/weather?q=copenhagen,dk&APPID=d28e0b6cfa6d48a373d2359ff966fbad&units=metric

URL til rejseplanens API:
https://xmlopen.rejseplanen.dk/bin/rest.exe/departureBoard?id=8600626&date=24.12.22&ti%20me=07:02&useBus=0

2.1.3 HttpGet

HTTP (Hypertext Transfer Protocol) anvendes til kommunikation på internettet, typisk involveres en klient enhed, som sender en "request" til en server, hvorefter serveren sender en "respons" besked tilbage. Når "repsons" er sendt tilbage, er forbindelsen mellem klient enheden og serveren udført og den ønskede information er hentet (Cloudfare, n.d.-a)

Der findes forskellige Http metoder. Vi har i vores rapport benyttet *HttpGet*. Denne metode anvendes til at hente informationer fra en server, uden at det påvirker anden data. Vi har benyttet os af denne metode, da vi i vores program henter informationer ned fra rejseplanen og vejrudsigten.

2.1.4 TCP + UDP

Transmission Control Protocol (TCP) og User Data Protocol (UDP). TCP og UDP befinder sig i "Transport Layer" i OSI-Modellen. "Transport Layer" repræsenterer end-to-end kommunikation mellem enheder der er forbundet til internettet (Cloudfare, n.d.-b). Det er i dette lag, hvor nedbrydning af data i segmenter, pakke data og samle data forekommer. UDP bekymrer sig hovedsageligt om at få tingene ud af dets eget netværk. Den kommunikerer ikke frem og tilbage og er derfor nyttig hvis man ønsker hurtighed, f.eks til live streams (Panigrahi, 2022). I UDP protokollen kræves det ikke at forbindelsen vedligeholdes, og derved sikring om at dataen er modtaget, og samlet.
TCP protokollen nedbryder data i nummeret pakker, håndtere fejl og sikre en pålidelig forbindelse mellem enhederne (Panigrahi, 2022). Med TCP er det ikke muligt at modtage halvdelen af dataen, men kan derimod enten modtage det hele eller intet (Panigrahi, 2022). Vi benytter TCP, da vi i højere grad finder det nyttigt at modtage alt data samt den korrekte data.

2.1.5 M15 - Pose Tracker + Key Points

Den implementeret "pose tracker" er machine learning, der estimerer menneskers kroppe, i realtid. Pose Trackeren identificerer forskellige kropsdele på mennesket og er bygget op af en algoritme, som estimerer forskellige punkter ud fra menneskets kropsled. Dette er baseret på machine learning. Den anvender machine learning metoden neural network, og er opbygget af neuroner, med vægte mellem hinanden og som lærer og genkender mønstre. I rapporten benyttes begrebet "keypoint", som er en kropsdel såsom næse, højre arm, venstre arm mv. Et keypoint indeholder både "position" og "confidence score" (Oved, 2018)

2.2 Opgavens funktioner

I dette afsnit beskrives udvalgte funktioner i programmet.

2.2.1 Globale url variabler og funktionen preload()

Nedenfor på "*Skærmbillede 1: Preload & url*", ses hvordan vi implementerer API'erne.

Først har vi lavet to globale variabler; *weather* og *urlWeather*. Værdien for variablen *urlWeather* er URLen for vejr API'en, det er også her vi angiver hvilken by vi ønsker at henter vejrinformationer omkring.

Ligeledes har vi lavet to globale variabler; *rejse* og *urlRejse*. Værdien for *urlRejse* er URL'en for Rejseplanens API.

I *function preload()* defineres gennem httpGet, hvilken url der anvendes. I httpGet body, sætter vi responset til vores globale variabel(*weather* og *rejse*), hvorefter vi anvender print med den globale variabel, til at printe i konsollen.

```
//URLs til api
var urlWeather = "https://api.openweathermap.org/data/2.5/weather?
q=copenhagen,dk&APPID=d28e0b6cfa6d48a373d2359ff966fbad&units=metric";

var urlRejse = "https://xmlopen.rejseplanen.dk/bin/rest.exe/departureBoard?
id=8600626&date=24.12.22&ti%20me=07:02&useBus=0"


// Et kald gennem httpGet til hver api i function preload.
function preload() {
  httpGet(urlWeather, "json", false, function (response) {
    weather = response;
    print(weather);
  });
     httpGet(urlRejse, "xml", false, function (response) {
    rejse = response;
    print(rejse);
  });
}
```

Skærmbillede 1: Preload & url

Konsollen vil derfor fremgå således se "*Skærmbillede 2: Konsol(print weather og rejse)*" forneden. Her ville vi kunne udfolde API'ens indhold, ved at trykke på de grå trekanter.

```
▶{coord: Object, weather: Array(1), base: "stations", main: Object, vis
ibility: 10000…}

▶{name: "DepartureBoard", attributes: Object, children: Array(41), pare
nt: null, content: ","…}
```

Skærmbillede 2: Konsol(print weather og rejse)

2.2.2 Funktionen setup()

Setup() funktionen kører en gang, når programmet startes. I denne funktion implementeres bl.a canvas, video og PoseNet. På nedenstående billede "*Skærmbillede 3: setup()*" ses hvordan machine learning PoseNet, som er vores Pose Tracker, implementeres.

```
79    //m15 har et neutral netværk,
80    //som er blevet trænet, som vi henter ned og anvender
81    poseNet = ml5.poseNet(video, poseOptions);
82
83    poseNet.on("pose", function (results) {
84      poses = results;
```

Skærmbillede 3: setup()

2.2.3 Funktionen draw()

Funktioner kaldt under draw() kører kontinuerligt og fortsætter indtil koden stoppes (p5.js, n.d.-a). Se nedenstående billede "*Skærmbillede 4: draw()*".
I denne funktion kaldes følgende funktioner; drawVideo(), drawPoses() og limpButton(). Det er muligt at bestemme hvor mange frames pr. sekund draw skal udføres, i vores program har vi bestemt 60 frames pr. sekund.

```
98  function draw() {
99    // make a black background
100   background(0);
101
102   // draws the video
103   drawVideo();
104   // draws the skeleton, the keypoints and the video
105   drawPoses();
106
107   //Henter temperaturen og rejsetider
108   limpButton(25,400,"SE REJSETIDER");
109   limpButton(300,400,"SE VEJRUDSIGTEN");
110 }
```

Skærmbillede 4: draw()

2.2.4 Funktionen drawVideo()

I funktionen drawVideo() programmeres selve video-displayet, ved image(video, 0,0). Dette kan ses på nedenstående billede "*Skærmbillede 5: drawVideo()*". Video-displayet er programmeret således, at det er et spejl for brugeren. Som default, er videoen spejlvendt for brugeren, hvilket denne funktion ændrer. Boolean flipVideo er sat til at være true, og hvis dette er sandt udføres if(flipVideo), således at videoen flippes og ikke er spejlvendt for brugeren. Før if-statement skaleres koordinatsystemet ved scale(), og derefter (i if-statement) anvendes translate(), som flytter koordinatsystemets (0,0) punkt. Derefter anvendes scale() igen til at flippe videoen (p5.js, n.d.-b; p5.js, n.d.-c).

```
127 function drawVideo() {
128   push();
129   scale(windowWidth / video.width);
130   if (flipVideo) {
131     //move image by the width of image to the left
132     translate(video.width, 0);
133     //then scale it by -1 in the x-axis
134     //to flip the image
135     scale(-1, 1);
136   }
137
138   image(video, 0, 0);
139   pop();
140 }
```

Skærmbillede 5: drawVideo()

2.2.5 Funktionen drawPoses()

I funktionen drawPoses() "tegnes" skelettet på brugeren. Denne funktion er hentet fra learn.hobye.dk (Hobye, u.d.). Der forekommer "nested" loops, da to for-loops er indeni det første for-loop. De første to for-loops kan ses på nedenstående billede "*Skærmbillede 6: For-loop skeleton*". Først løbes alle identificeret skeletter igennem, derefter løbes alle forbindelserne igennem for hvert skelet.

```
// Loop through all the skeletons detected
for (let i = 0; i < min(1, poses.length); i++) {
  stroke(255, 255, 255, 220);
  // For every skeleton, loop through all body connections
  for (let j = 0; j < poses[i].skeleton.length; j++) {
    let partA = poses[i].skeleton[j][0];
    let partB = poses[i].skeleton[j][1];
    line(
      partA.position.x,
      partA.position.y,
      partB.position.x,
      partB.position.y
    );
  }

  fill(255, 255, 255, 50);
  noStroke();
```

Skærmbillede 6: For-loop skeleton

Når alle forbindelserne er løbet igennem, løbes alle keypoints igennem, for hvert identificeret pose, hvilket kan ses på nedenstående billede "*Skærmbillede 7: Loop Keypoints*". Derefter tegnes en ellipse, ved disse keypoints. Dette kan ses ved nedenstående billeder "*Skærmbillede 8: Ellipse*"

```
    // For each pose detected, loop through all the keypoints
    for (let j = 0; j < poses[i].pose.keypoints.length; j++) {
      // A keypoint is an object describing a body part (like
rightArm or leftShoulder)
      let keypoint = poses[i].pose.keypoints[j];
```

Skærmbillede 7: Loop Keypoints

```
ellipse(keypoint.position.x, keypoint.position.y, 10, 10);
text(j, keypoint.position.x, keypoint.position.y);
```

Skærmbillede 8: Ellipse

2.2.6 Funktionen gotPoses(results)

Funktionen gotPoses(results), kan ses på nedenstående billede "*Skærmbillede 5: gotPoses()*". Denne funktion tager variablen results som parameter, og kaldes hver gang der er en opdatering fra PoseNet modellen.

```
183 function gotPoses(results) {
184   poses = results;
185 }
```

Skærmbillede 9: gotPoses()

2.2.7 Funktionen limpButton(x,y,tekst)

Funktionen limpButton tager 3 paramete; x, y og tekst. Overordnet indeholder denne funktion 4 forskellige blokke af kode. Den første "tegner" boksene; Se rejsetider og Se vejrudsigten. Det er disse bokse som brugeren skal forsøge at ramme. Til at positionere disse anvendes x og y fra funktionens parameter. Ydermere "tegnes" et tekstfelt, som udfyldes med variablen tekst, angivet i funktionens parameter. Tekstfeltet placeres ovenpå boksene. Dette refereres til "knapper" i denne rapport.

Derefter forekommer der 2 if-statements som implementerer hvor længe rejse- og vejr informationerne skal vises, når brugeren rammer knapperne. Dette ses på nedenstående billede "*Skærmbillede 10: Timer*".

```
if (rejseKnapState == true ){
  if(millis() - rejseTimer > 10000){
    rejseTimer = millis();
    rejseKnapState = false;
   // console.log("rejseHit");

  } else{
 //console.log(rejseKnapState);
    getRejse();
  }
    }

if(vejrKnapState == true){
  if(millis() - vejrTimer > 10000){
    vejrTimer = millis();
    vejrKnapState = false;
   //console.log("vejrHit");
  } else {
  //console.log(vejrKnapState);
    getTemp();
  }
}
```

Skærmbillede 10: Timer

I begge if-statements skal følgende globale booleans "rejseKnapState" og "vejrKnapState" være "true". Derefter tjekkes hvorvidt millis() - rejseTimer er større end 10000, hvis dette også er sandt. Skal "rejseKnapState" og "vejrKnapState" sættes til false, og derved er den første if-statement ikke længere sand. Hvis den ikke er sand, kaldes funktionerne getRejse og/eller getTemp.

Ydermere forekommer der en blok kode, hvor getRejse og getTemp kaldes, hvis brugerens håndled er indenfor knapperne. Dette ses på nedenstående billede "*Skærmbillede 11: LimpPosition indenfor knapper*".

```
if ((getLimpPosition(0, 10).x > x
     && getLimpPosition(0, 10).x < x+250

     && getLimpPosition(0, 10).y > y
     && getLimpPosition(0, 10).y < y + 50)

    || (getLimpPosition(0, 9).x > x
        && getLimpPosition(0, 9).x < x+250
        && getLimpPosition(0, 9).y > y
        && getLimpPosition(0, 9).y < y + 50)) {
```

Skærmbillede 11: LimpPosition indenfor knapper

Denne if-statement sammenligner først positionen (x og y) for keypoint 10 med variablerne x og y. X positionen for keypoint 10 skal være større end x og mindre end x+250. Y positionen skal være mindre end y og større end y + 50. Ligeledes sammenlignes førnævnte blot med keypoint 9. Der forekommer en "or" operator, hvor det enten er den ene side eller den anden side af "or" som skal være sand, hvis kode indenfor if-statement skal udføres. X og Y pluses

med 250 og 50, da det er højden og bredden for knapperne. Det er derved muligt at vurdere om keypoint 10 og 9 er indenfor knapperne og ikke blot rammer et punkt på x og y aksen.

Hvis ovenstående er sandt, udføres en blok kode som bestemmer hvorvidt det er rejse eller vejr informationer der skal vises. Dette kan ses på nedenstående billede "*Skærmbillede 12: Hvilken information*". Kalder funktionerne getRejse og getTemp i henhold til hvilket knap keypoint 10 og/eller 9 er over.

```
      //Hvis teksten er se rejsetider, kalder den "getRejse(); "
      if (tekst == 'SE REJSETIDER' ){
        getRejse();
        rejseKnapState = true; //Sættes til true, anvendes i linje
198 i   if-statement til at sætte timer.
      }

      //Hvis teksten er se vejrudsigten, kalder den "getTemp();"
      if (tekst == 'SE VEJRUDSIGTEN'){
        getTemp();
        vejrKnapState = true; //Sættes til true, anvendes i linje
210 i if-statement til at sætte timer.
      }
    }
  }
}
```

Skærmbillede 12: Hvilken information

2.2.8 Funktionen getLimpPosition(person, id)

Funktionen getLimpPosition henter positionerne af limps x og y koordinater. Der oprettes en lokal variabel *position, som* henter keypossitions af posene, af personen i billedrammen. Variablen "mapped" sætter x og y koordinaterne til 0, hvorefter at mapped x og y defineres. Mapped.x = Map() går ind og omformer et tal fra et område til et andet og er nyttig når man skal skalere sine elementer. Funktionen returnerer mapped; og vi får i denne funktion den givne position på personens poses.

```
115 function getLimpPosition(person, id) {
116   var position = poses[person].pose.keypoints[id].position;
117
118   var mapped = {
119     x: 0,
120     y: 0,
121   };
122   var scale = windowWidth / video.width;
123   mapped.x = map(position.x, 0, video.width, 0, windowWidth);
124   mapped.y = position.y * scale;
125   return mapped;
126 }
```

Skærmbillede 13: getLimpPosition(person, id)

2.2.9 Funktionen getRejse()

Funktionen getRejse tilgås den ønskede information fra rejseplanens API. API'en henter "DepartureBoard", hvor der forekommer 41 arrays (under punkt: Children). Disse arrays indeholder data om afgange fra København H. Et array kan også være undefined og derved ikke indeholde data. Hvis arrayet indeholder data, ses dette under "Attributes" under det valgte array og tilgås ved rejse.children[indeks].attributes.DATA. Informationerne som printes er lavet i et forloop, hvilket kan ses på nedenstående billede "*Skærmbillede 14: getRejse()*".

```
function getRejse(){
  var xRect = 25;
  var yRect = 10;
  var wRect = windowWidth/2;
  var hRect = windowHeight-160;

       textSize(10);
      textAlign(LEFT);
      fill(222,234,255);
      rect(xRect, yRect, wRect, hRect);
      fill(198,34,216);

  yPlacering = 0;
for(var i = 0; i<12; i++){
  if(rejse.children[i] != undefined){
    if(rejse.children[i].attributes.time != undefined){

  text("\n Tidspunkt: " + rejse.children[i].attributes.time +"\n Afgang fra: "
+ rejse.children[i].attributes.stop + "\n Mod: " +
rejse.children[i].attributes.finalStop + "\n Togtype: " +
rejse.children[i].attributes.type,xRect,yRect + yPlacering );

  yPlacering += 55; //tilføjer 50, for at rykke y-position, således at teksten
ikke bliver placeret samme sted.
```

Skærmbillede 14: getRejse()

For-loopet køres så længe i er mindre end 12. I for-loopet er der implementeret to if-statements, som er sande hvis rejse.children[i] (indeks i) ikke er undefined og hvor attributten "time" hellere ikke er undefined. Hvis disse to er sande, udføres text() funktionen, hvori data om indeksets tid, stop og finalStop og type indsættes i tekstfeltet. Derudover øges yPlaceringen med 55, således at tekstfeltet printes med 55 til forskel på y-aksen.

2.2.10 Funktionen getTemp()

Funktionen getTemp() tilgår den ønskede funktion, hvilket kan ses på nedenstående billede "*Skærmbillede 15: getTemp()*". Der forekommer en if-statement, som er sand hvis variablen weather ikke er undefined. Derefter tilgås data om temperaturen, feels_like, maks temperaturen og humidity.

```
function getTemp(){
  var xRect = 522;
  var yRect = 110;
  var wRect = 100;
  var hRect = 70;

  textSize(10);
  textAlign(CENTER);


  fill(222,234,255);
      rect(xRect, yRect, wRect, hRect);
      fill(198,34,216);

    if (weather != undefined) {
      text(weather.name + "\n Temperatur: "+ weather.main.temp + "\n
Føles som: " + weather.main.feels_like + "\n Max temperatur: " +
weather.main.temp_max + "\n Luftfugtighed: " + weather.main.humidity,
570, 120);
```

Skærmbillede 15: getTemp()

Derudover, har vi implementeret at der ved temperatur informationerne vises et billede af enten en sky eller en sol. Dette ses på nedenstående billede "*Skærmbillede 16: getTemp() billeder*". Her vises et billede af en sky, hvis weather.main.temp er mindre end eller lig 10 grad, og hvis temp er større end 10 vises et billede af en sol.

```
      //Hvis temperaturen er 10 eller mindre, vil der fremgå et
billede af en sky.
     if(weather.main.temp <= 10 ){
       image(cloud, 522, 5, 100,125);
     }

      //Hvis temperaturen er over 10, vil der fremgå et billede af
en sol.
     if(weather.main.temp > 10 ){
        image (sun,522,5,100,100);

     }
  }
```

Skærmbillede 16: getTemp() billeder

Konklusion

Problemet vi havde til formål at løse, var en optimering af de nuværende informationsskærme på stoppesteder og stationer. Denne løsning medfører at man undgår touch skærme og med få poseringer kan få nyttige informationer. En videreudvikling af interaktive produkt kunne være at tilføje vejret for den lokation man rejser til. Ydermere kunne man implementere et live kort, som viser togene komme ind på perronen eller om de allerede holder der.

Litteraturliste

Cloudfare. (n.d.). What is HTTP. Cloudfare. Tilgået November 21, 2022, fra https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/

Cloudfare. (n.d.). What is the OSI Model? Cloudfare. Tilgået November 21, 2022, fra https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/

Hobye, M. (n.d.). Machine Learning. Learn.Hobye.Dk. Tilgået November 21, 2022, fra https://learn.hobye.dk/kits/machine-learning

Oved, D. (2018, May 7). Real-time Human Pose Estimation in the Browser with TensorFlow.js. Medium. Tilgået November 21 fra https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5

p5.js. (n.d.-a). Reference: Draw(). P5js. Tilgået November 21, 2022, fra https://p5js.org/reference/#/p5/draw

p5.js. (n.d.-b). Reference: Scale(). P5js. Tilgået November 21, 2022, fra https://p5js.org/reference/#/p5/scale

p5.js. (n.d.-c). Reference: Translate(). P5js. Tilgået November 21, 2022, fra https://p5js.org/reference/#/p5/translate

TutorialsPoint. (n.d.). Difference between TCP and UDP. TutorialsPoint. Tilgået November 21, 2022, fra https://www.tutorialspoint.com/difference-between-tcp-and-udp

Difference between JSON and XML. (u.d.). Hentet fra Geeks for Geeks: https://www.geeksforgeeks.org/difference-between-json-and-xml/

Hobye, M. (u.d.). Learn.Hobye.dk. Hentet fra Machine Learning: https://learn.hobye.dk/kits/machine-learning

JSON. (u.d.). Hentet fra Wikipedia: https://en.wikipedia.org/wiki/JSON

Panigrahi, K. K. (11. August 2022). Difference between TCP and UDP. Hentet fra Tutorialspoint: https://www.tutorialspoint.com/difference-between-tcp-and-udp

REST API Architectural Constraints. (u.d.). Hentet fra Geeks for Geeks: https://www.geeksforgeeks.org/rest-api-architectural-constraints/?ref=lbp

What is a URL? - Learn web development. (n.d.). MDN. Retrieved November 24, 2022, from https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL