Facultatea de Electronică, Telecomunicaţii şi TI

Proiect informatica aplicata

Coordonator:
Pr. Sabau Gabriel

Studenti:
Nistor Flaviu-Cristian
Marinescu Silviu-Andrei

Cuprins

1. Introducere

Programarea este aranjarea în ordine cronologică a mișcărilor, operațiilor, acțiunilor sau activităților cu scopul de a obține o stare dorită a unui sistem în finalul unei perioade. De obicei, programarea este asociată cu activitatea umană, dar există semne că și în natură pot exista forme de programare, cum ar fi aranjarea proceselor genetice sau comportamentele instinctuale ale animalelor.

Programarea calculatorului este o activitate informatică care implică dezvoltarea de produse-program și programe (software) necesare pentru a realiza diverse activități cu ajutorul unui calculator. Această activitate include etapele de specificare, proiectare, implementare, documentare și întreținere a produsului program.

Termenul "informatică" se referă la știința procesării sistematice a informației, în special prin utilizarea calculatoarelor. Corespondentul în limba engleză este "Computer Science". Din punct de vedere istoric, informatica a evoluat ca o știință derivată din matematică, în timp ce primele calculatoare au fost dezvoltate în domeniul electrotehnicii și telecomunicațiilor. De aceea, calculatorul este considerat doar dispozitivul pe care se implementează conceptele teoretice. După cum spunea informaticianul olandez Edsger Dijkstra: "În informatică, lucrezi cu calculatorul așa cum în astronomie lucrezi cu telescopul".

Nu trebuie confundată informatica cu tehnologia informației sau cu teoria informației. Informatica se divide în trei domenii fundamentale: informatică teoretică, informatică aplicată și informatică tehnică. Pe lângă acestea, există și domenii suplimentare, cum ar fi inteligența artificială, considerată interdisciplinară și într-o anumită măsură autonomă. Utilizarea informaticii în diverse domenii ale vieții de zi cu zi, precum economie, geografie și domeniul medical, este cuprinsă în ceea ce se numește informatică aplicată.

Informatica teoretică reprezintă baza pentru alte domenii derivate. Aceasta furnizează cunoștințe fundamentale privind decidabilitatea problemelor, sistematizarea complexității și formalizarea automatelor și limbajelor formale.

Pe baza acestor fundamente teoretice se dezvoltă informatica practică și informatica tehnică, care se ocupă de problemele centrale ale prelucrării informației și oferă soluții practice și adaptabile. Aceste două domenii sunt strâns legate, diferențiindu-se în funcție de gradul de apropiere sau de departare de microelectronică. Din perspectiva informaticii, electronica reprezintă doar un instrument și nu un domeniu central de cercetare. În informatica practică, se caută soluții care să minimizeze dependența de electronică.

Rezultatele obținute în aceste domenii găsesc aplicații practice în informatica aplicată. Acest domeniu se ocupă de realizarea echipamentelor și programelor (hardware și software), iar o mare parte a pieței tehnologiei informației (IT) este acoperită de acesta. În domeniile interdisciplinare se desfășoară cercetări pentru a găsi soluții posibile oferite de tehnologia informației, cum ar fi dezvoltarea sistemelor geoinformaționale, informatica economică sau bioinformatica.

2. Considerente teoretice

2.1. Modulul ESP32

Modulul ESP32 este un modul SoC (System on Chip) fabricat de compania Espressif Systems, bazat pe microprocesorul Tensilica Xtensa LX6 cu unul sau două nuclee și o frecvență de lucru de între 160 și 240MHz precum și un coprocesor ULP (Ultra Low Power). Suplimentar, acesta dispune de comunicație WiFi și Bluetooth (clasic și low-energy) integrate, precum și de o gamă largă de interfețe periferice:

- 34 pini programabili GPIO (General Purpose Input/Output)
- 18 canale de conversie analog-digitală (ADC) cu rezoluție de 12 biți
- 2 canale de conversie digital-analogică (DAC) cu rezoluție de 8 biți
- 16 canale de ieșire PWM (Pulse Width Modulation)
- 10 senzori interni capacitive
- 3 interfețe SPI (Serial Peripheral Interface)
- 3 interfețe UART (Universal Asynchronous Receiver-Transmitter)
- 2 interfețe I2C (Inter-Integrated Circuit)
- 2 interfețe I2 S (Inter-IC Sound)
- 1 interfață CAN 2.0 (Controller Area Network)
- controler pentru conectarea dispozitivelor de stocare (carduri de memorie)

Modulul ESP32 poate fi integrat în plăci de dezvoltare ce pot expune toți pinii/interfețele modulului sau doar o parte din ele. Cele mai des întâlnite tipuri de plăci de dezvoltare bazate pe modulul ESP32 sunt cele cu 30 sau 38 de pini. În Fig. 1 este prezentată diagrama unei plăci de dezvoltare cu 38 de pini, placă ce va fi folosită în cadrul acestui proiect. Numerotarea pinilor de pe placa de dezvoltare este realizată în funcție de denumirea internă a pinilor modulului (GPIOxx). Placa dispune de un LED integrat care este conectat la pinul general de intrare/ieșire GPIO02, care poate fi accesat pe placă pe pinul G2 (Fig. 1).

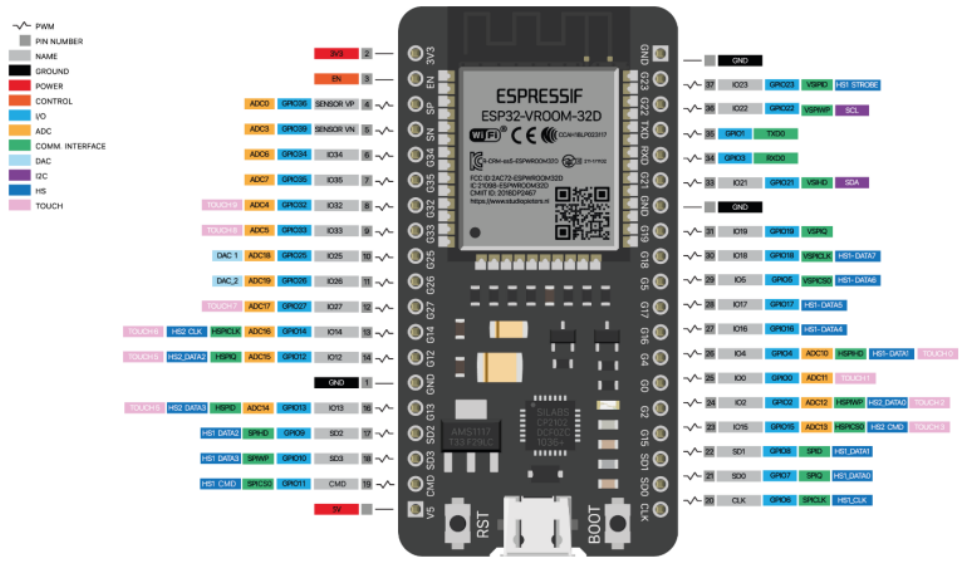


Fig. 1 - Diagrama pinilor plăcii de dezvoltare ESP32 – varianta cu 38 de pini [1]

2.2. Protocoalele si metodele de comunicatie utilizate

PROTOCOALE

Un protocol de comunicare reprezintă un sistem de reguli care permite transmiterea informațiilor între două sau mai multe entități în cadrul unui sistem de comunicații prin intermediul unei variații a unei cantități fizice. Protocolul definește regulile, sintaxa, semantica și sincronizarea comunicării, precum și metodele posibile de gestionare a erorilor. Protocoalele pot fi implementate prin intermediul componentelor hardware, software sau prin utilizarea unei combinații a acestora.

Sistemele de comunicare utilizează formate bine definite pentru schimbul de mesaje diverse. Fiecare mesaj are o semnificație precisă și are scopul de a obține un răspuns dintr-o serie de răspunsuri posibile predefinite pentru situația respectivă. Comportamentul specificat în protocol este, de obicei, independent de modalitatea în care este implementat. Protocoalele de comunicare trebuie convenite și adoptate de către părțile implicate în comunicare. Pentru a atinge un acord, un protocol poate fi dezvoltat într-un standard tehnic. Există o analogie strânsă între protocoalele de comunicare și limbajele de programare: protocoalele sunt pentru comunicare ceea ce limbajele de programare sunt pentru calcule. O formulare alternativă afirmă că protocoalele sunt pentru comunicare ceea ce algoritmii sunt pentru calcul.

De multe ori, diferite aspecte ale unei comunicări sunt descrise de mai multe protocoale distincte. Un grup de protocoale concepute pentru a colabora împreună formează o suită de protocoale; atunci când sunt implementate în software, acestea formează o stivă de protocoale.

Protocoalele de comunicare pentru internet sunt publicate de Internet Engineering Task Force (IETF). IEEE (Institutul de Ingineri Electrici și Electronici) se ocupă de rețelele cu fir și fără fir, în timp ce Organizația Internațională pentru Standardizare (ISO) gestionează alte tipuri de protocoale. ITU-T (Uniunea Internațională de Telecomunicații - Sectorul pentru Standardele de Telecomunicații) administrează protocoalele și formatele de telecomunicații pentru rețelele publice de telefonie comutată (PSTN). Pe măsură ce PSTN și internetul converg, se urmărește și o convergență a standardelor.

Concept

Informațiile care sunt schimbate între dispozitive prin intermediul unei rețele sau altor medii sunt supuse regulilor și convențiilor stabilite în specificațiile protocolului de comunicație. Aceste specificații definesc natura comunicării, datele efectiv schimbate și orice comportament care depinde de starea sistemului. În sistemele de calcul digital, regulile pot fi exprimate prin intermediul algoritmilor și structurilor de date. Astfel, protocoalele sunt pentru comunicare ceea ce algoritmii sau limbajele de programare sunt pentru calcule.

În sistemele de operare, de obicei există un set de procese cooperante care manipulează datele partajate pentru a comunica între ele. Această comunicare este guvernată de protocoale bine definite, care pot fi incorporate în codul procesului însuși. Pe de altă parte, în sistemele de comunicare, deoarece nu există memorie partajată, sistemele trebuie să comunice între ele folosind un mediu de transmisie partajat. Transmisia poate fi nesigură, iar sistemele individuale pot utiliza hardware sau sisteme de operare diferite.

Pentru a implementa un protocol de rețea, modulele software de protocol sunt interfațate cu un cadru implementat pe sistemul de operare al mașinii. Acest cadru implementează funcționalitatea de rețea a sistemului de operare. Atunci când algoritmii de protocol sunt exprimați într-un limbaj de programare portabil, software-ul de protocol poate deveni independent de sistemul de operare. Cele mai cunoscute cadre sunt modelul TCP/IP și modelul OSI.

În momentul în care a fost dezvoltat Internetul, s-a dovedit că abordarea stratificării abstracției este o metodă de proiectare de succes atât pentru compilatoare, cât și pentru sistemele de operare. Având în vedere asemănările dintre limbajele de programare și protocoalele de comunicare, programele inițial monolitice de rețea au fost dezmembrate în protocoale cooperante. Acest lucru a condus la conceptul de protocoale stratificate care formează acum baza designului protocolului.

Protocoalele pot fi organizate în grupuri pe baza funcționalității, de exemplu, există un grup de protocoale de transport. Funcționalitățile sunt mapate pe straturi, fiecare strat rezolvând o clasă distinctă de probleme, cum ar fi funcțiile de aplicație, transport, internet și interfață de rețea. Pentru a transmite un mesaj, trebuie selectat un protocol din fiecare strat. Se selectează următorul protocol prin extinderea mesajului cu un selector de protocol pentru fiecare strat.

Tipuri

Există două tipuri de protocoale de comunicare, care diferențiază în funcție de modul în care reprezintă conținutul transportat: protocoale bazate pe text și protocoale bazate pe binar.

Protocoalele bazate pe text utilizează reprezentări textuale pentru a transmite informații între dispozitive. Acestea pot folosi formate de date precum JSON (JavaScript Object Notation), XML (eXtensible Markup Language) sau chiar formatul text simplu. Avantajul protocoalelor bazate pe text constă în faptul că sunt ușor de citit și de interpretat de către oameni, ceea ce facilitează dezvoltarea, depanarea și interoperabilitatea între diferite dispozitive sau sisteme.

Pe de altă parte, protocoalele bazate pe binar utilizează reprezentări binare pentru a transmite informațiile. Acestea codifică datele într-un format compact, optimizat pentru eficiența transferului de date între dispozitive. Protocoalele bazate pe binar pot folosi structuri de date binare, cum ar fi octeți și biți, pentru a reprezenta informațiile. Acest tip de protocoale este adesea utilizat în aplicații care necesită o transmitere rapidă și eficientă a datelor, cum ar fi transferul de fișiere sau comunicațiile în timp real.

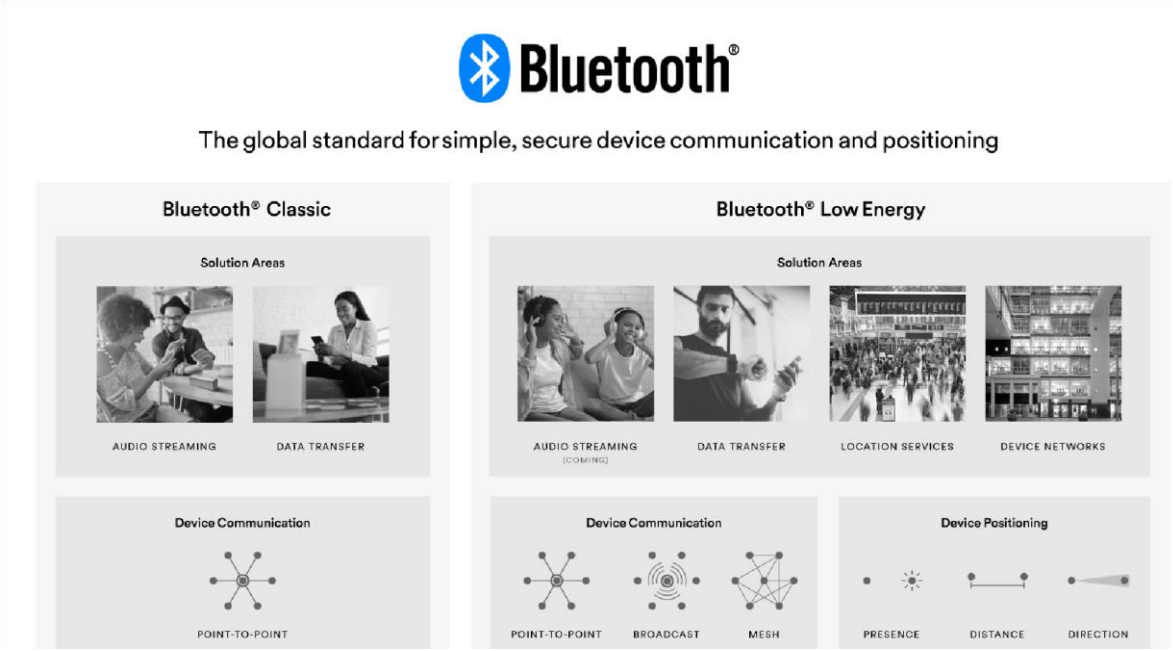
Alegerea între un protocol bazat pe text sau unul bazat pe binar depinde de necesitățile aplicației și de resursele disponibile. Protocoalele bazate pe text sunt mai ușor de citit și de interpretat, dar pot necesita o cantitate mai mare de date pentru a transmite aceeași informație în comparație cu protocoalele bazate pe binar. Pe de altă parte, protocoalele bazate pe binar pot oferi o transmitere mai eficientă a datelor, dar pot necesita o etapă suplimentară de interpretare a informațiilor la nivelul destinatarului.

Tehnologie fără fir Bluetooth

Standardul de schimb de date fără fir Bluetooth utilizează o varietate de protocoale pentru a facilita comunicarea între dispozitive. Protocoalele de bază sunt definite de organizația comercială Bluetooth SIG (Special Interest Group), iar protocoale suplimentare de la alte organisme de standardizare au fost adoptate. În continuare, vom prezenta o privire de ansamblu asupra protocoalelor de bază și a celor adoptate pe scară largă.

Stiva de protocol Bluetooth este împărțită în două părți principale: o "stivă de controler" care conține interfața radio critică pentru sincronizare și o "stivă gazdă" care se ocupă de datele de nivel superior. Stiva de controler este de obicei implementată într-un dispozitiv de siliciu cu costuri reduse, care include un radio Bluetooth și un microprocesor. Stiva gazdă, pe de altă parte, este de obicei implementată ca parte a unui sistem de operare sau ca un pachet instalabil pe un sistem de operare existent. În cazul dispozitivelor integrate, cum ar fi căștile Bluetooth, stiva gazdă și stiva controlerului pot rula pe același microprocesor pentru a reduce costurile de producție în masă; acest tip de implementare este cunoscut sub numele de "sistem fără gazdă".

Unul dintre motivele cheie pentru succesul tehnologiei Bluetooth este flexibilitatea extraordinară pe care o oferă dezvoltatorilor. Prin furnizarea a două opțiuni radio, tehnologia Bluetooth oferă dezvoltatorilor un set versatil de soluții complete, potrivite pentru a satisface nevoile în continuă expansiune ale conectivității wireless. Indiferent dacă un produs transmite în flux audio de înaltă calitate între un smartphone și un difuzor, transferă date între o tabletă și un dispozitiv medical sau trimite mesaje între mii de noduri într-o soluție de automatizare a clădirilor, radiourile Bluetooth Low Energy (LE) și Bluetooth Classic sunt concepute pentru a satisface nevoile unice ale dezvoltatorilor din întreaga lume.



Bluetooth® LOW ENERGY

Tehnologia Bluetooth Low Energy (LE) oferă dezvoltatorilor o soluție completă și adaptată pentru a satisface nevoile lor specifice în domeniul conectivității wireless cu consum redus de energie. Bluetooth LE este optimizat pentru aplicații care necesită o durată lungă de viață a bateriei și utilizează un consum redus de energie în timpul comunicațiilor. Acest tip de radiou Bluetooth este ideal pentru dispozitive precum dispozitivele purtabile, senzorii de mediu, dispozitivele medicale și alte dispozitive care necesită o interacțiune constantă cu utilizatorul sau transmiterea periodică a datelor. Prin intermediul tehnologiei Bluetooth LE, dezvoltatorii beneficiază de avantajele conectivității wireless eficiente din punct de vedere energetic și pot crea produse inovatoare și fiabile într-o gamă largă de domenii, inclusiv sănătate și fitness, automatizare a clădirilor, Internetul lucrurilor (IoT) și multe altele.

Bluetooth® Classic

Tehnologia Bluetooth Classic oferă dezvoltatorilor o soluție versatilă și robustă pentru a satisface nevoile lor în ceea ce privește conectivitatea wireless. Bluetooth Classic este optimizat pentru transferul de date de mare viteză și pentru transmiterea de semnal audio de înaltă calitate între dispozitive. Acest tip de radiou Bluetooth este ideal pentru aplicații precum streaming-ul de muzică în timp real între dispozitive audio, conectarea dispozitivelor mobile cu sistemele de divertisment ale vehiculelor sau transferul rapid de fișiere între dispozitive. Cu tehnologia Bluetooth Classic, dezvoltatorii pot crea produse cu performanțe excelente în ceea ce privește transmiterea datelor și conectivitatea între dispozitive, satisfăcând astfel cerințele din industrii precum audio, telecomunicații, gaming și multe altele.

WIFI

Wi-Fi, cunoscut și sub denumirea de WLAN (rețea locală fără fir), este o tehnologie de comunicație bazată pe standardele IEEE 802.11. Aceasta permite crearea de rețele locale de comunicație fără fir cu viteze comparabile cu cele ale rețelelor cu fir Ethernet. Dispozitivele hardware și sistemele de operare moderne pentru calculatoare personale, rutere, telefoane mobile, console de jocuri și televizoare avansate oferă suport pentru Wi-Fi.

Standardul IEEE 802.11 definește protocoale de comunicare la nivelul gazdă-rețea al Modelului TCP/IP și la nivelurile fizic și de legătură de date ale Modelului OSI. Acest lucru înseamnă că implementările IEEE 802.11 trebuie să primească pachete de la protocoalele de la nivelul rețea (IP) și să le transmită, evitând eventualele coliziuni cu alte stații care doresc să transmită.

Standardul 802.11 face parte dintr-o familie de standarde elaborate de IEEE pentru rețele locale, inclusiv standardul 802.3 pentru Ethernet. În timp ce Ethernet câștiga popularitate în anii '90, s-au făcut eforturi pentru ca noul standard să fie compatibil cu acesta în ceea ce privește transmiterea pachetelor.

IEEE a elaborat standardul în anii '90, iar prima versiune a fost finalizată în 1997. Implementatorii nu mai utilizează acea versiune, ci versiunile mai noi și îmbunătățite 802.11a/b/g, care au fost publicate între 1999 și 2001. Din 2004, se lucrează la versiunea 802.11n, care, deși nu a fost finalizată, este deja implementată de unii furnizori de echipamente.

În ceea ce privește securitatea, IEEE și Wi-Fi Alliance recomandă utilizarea standardului de securitate 802.11i, cunoscut și sub denumirea de WPA2. Alte metode simple de control al accesului într-o rețea 802.11 sunt considerate nesigure, precum schema WEP, care se bazează pe algoritmul de criptare simetrică RC4, cunoscut pentru vulnerabilitățile sale.

Limitările standardului provin din mediul fără fir utilizat, care face ca rețelele IEEE 802.11 să fie mai lente decât cele cu fir, precum Ethernet, și din utilizarea benzii de frecvență de 2,4 GHz, împărțită în 12 canale care se suprapun parțial. Consumul mare de energie și reglementările privind puterea electromagnetică emisă limitează raza de acoperire la câteva sute de metri, iar mobilitatea în cadrul acestor rețele este restrânsă. Cu toate acestea, au apărut tehnologii care permit conexiuni Wi-Fi între două puncte fixe situate la distanțe de sute de kilometri, depășind astfel limitările tradiționale.

PROTOCOLUL HTTP

HTTP (Hypertext Transfer Protocol) reprezintă metoda predominantă de accesare a informațiilor pe Internet, care sunt stocate pe serverele World Wide Web (WWW).

Protocolul HTTP este un protocol de tip text și este protocolul implicit al WWW.

Dacă un URL nu conține o parte de protocol, se presupune că acesta este de tip HTTP.

HTTP presupune că pe computerul destinație rulează un program care înțelege protocolul.

Protocolul HTTP este clasificat ca un protocol de nivel aplicație conform modelului de referință OSI.

Realizarea și evoluția acestuia sunt coordonate de către World Wide Web Consortium (W3C).

Modul de funcționare:

Atunci când se accesează un link sau o adresă web, cum ar fi http://adresa-server, se solicită computerului host să afișeze o pagină web (de exemplu, index.html).

În primă fază, numele (adresa) adresa-server este convertit de protocolul DNS într-o adresă IP.

Urmează transferul prin protocolul TCP pe portul standard 80 al serverului HTTP, în răspuns la cererea HTTP-GET.

Informații suplimentare, precum indicații pentru browser sau limba dorită, pot fi adăugate în antetul pachetului HTTP.

În urma cererii HTTP-GET, serverul răspunde cu datele solicitate, cum ar fi pagini în format (X)HTML, imagini atașate, fișiere de stil (CSS) sau scripturi (JavaScript). De asemenea, pot fi generate și pagini dinamice (folosind SSI, JSP, PHP și ASP.NET).

În cazul în care informațiile nu pot fi transmise dintr-un motiv anume, serverul trimite un mesaj de eroare conform specificațiilor HTTP.

Metodele disponibile în HTTP sunt:

- GET: cea mai utilizată metodă, folosită atunci când serverul primește o cerere pentru o resursă.

- HEAD: similară cu metoda GET, dar serverul returnează doar antetul resursei, fără corpul acesteia.

- PUT: utilizată pentru a încărca documente pe server.

- POST: proiectată pentru a trimite date de intrare către server.

- DELETE: opusul metodei PUT, folosită pentru ștergerea unei resurse.

- TRACE: utilizată în scopuri de diagnosticare, oferind informații despre traseul urmat de legătura HTTP.

- OPTIONS: folosită pentru a identifica capacitățile serverului web înainte de a face o cerere.

- CONNECT: utilizată în principal de serverele intermediare.

Versiunile protocolului HTTP sunt:

- HTTP/0.9: prima versiune, realizată de Tim Berners-Lee și echipa sa, foarte simplă, dar cu multe neajunsuri, fiind rapid înlocuită.

- HTTP/1.0: versiune introdusă în 1996 prin RFC1945, aducând numeroase îmbunătățiri.

- HTTP/1.1: versiune de îmbunătățire și corectare a neajunsurilor versiunilor anterioare. În prezent, se utilizează două versiuni, HTTP/1.0 și HTTP/1.1.

- HTTP/1.0 stabilește o nouă conexiune TCP înaintea cererii, iar după transmiterea răspunsului, conexiunea trebuie închisă. Astfel, pentru un document HTML care conține 10 imagini, vor fi necesare 11 conexiuni TCP pentru a afișa complet pagina în browser.

- HTTP/1.1 permite emiterea mai multor cereri și răspunsuri pe aceeași conexiune TCP. Astfel, pentru același document HTML cu 10 imagini, este necesară doar o singură conexiune TCP. Algoritmul Slow-Start, care inițializază viteza conexiunii TCP la un nivel scăzut, este necesar doar o singură dată, ceea ce reduce semnificativ durata totală de încărcare a paginii. De asemenea, versiunea 1.1 poate relua și continua transferurile întrerupte.

Erorile HTTP sunt clasificate în 5 categorii:

- 1xx: erori informaționale, indicând un răspuns provizoriu al serverului.

- 2xx: răspunsuri reușite, indicând că cererea a fost primită, înțeleasă și acceptată cu succes.

- 3xx: redirectări, indicând că browser-ul trebuie să efectueze acțiuni suplimentare pentru a îndeplini cererea.

- 4xx: erori ale utilizatorilor, semnalând greșeli în formularea cererii. Serverul poate include explicații și indica dacă eroarea este temporară sau permanentă.

- 5xx: erori de server, indicând că serverul conștientizează greșelile sau este incapabil să execute cererea. Serverul ar trebui să includă o explicație a situației de eroare și să specifice dacă este temporară sau permanentă, cu excepția cererilor de tip redirectare.

JSON

JSON, cunoscut și ca JavaScript Object Notation, este un format text utilizat pentru reprezentarea și interschimbul de date între aplicații informatice. Este un format ușor de înțeles pentru oameni și este utilizat în special pentru transmiterea datelor structurate prin rețea, prin procesul numit serializare. JSON oferă o alternativă mai simplă și mai facilă decât limbajul XML. Acest format beneficiază de eleganța faptului că este un subset al limbajului JavaScript (ECMA-262 3rd Edition) și este adesea utilizat împreună cu acest limbaj. JSON a fost creat de Douglas Crockford și standardizat prin RFC 4627. Documentele JSON trebuie să fie transmise cu tipul de media "application/json" și au extensia de fișier ".json".

JSON utilizează o structură de date încuibată similară cu XML pentru a reprezenta informațiile. De asemenea, este un format text, codat în Unicode. JSON poate fi citit relativ ușor de către oameni, având o inteligență similară cu XML. Principala diferență constă în dimensiunea datelor: JSON utilizează mult mai puțină redundanță în reprezentare. Simplitatea formatului JSON, bazat pe JavaScript, îl face ideal pentru utilizarea împreună cu acest limbaj, dar nu este limitat doar la JavaScript.

Datorită avantajelor sale, JSON a câștigat rapid popularitate și au fost create implementări de analizoare JSON în aproape toate limbajele de programare existente, inclusiv C++, C#, Java și chiar limbaje mai exotice precum Limbo, Ruby și Smalltalk. JSON a devenit un instrument foarte util pentru transferul de date între diferite limbaje de programare, un exemplu notabil fiind utilizarea sa în tehnologia AJAX. Comparativ cu XML, JSON facilitează transmiterea asincronă a informațiilor între server și client. JSON este un subset al limbajului JavaScript și este construit în principal din două structuri: obiecte (perechi nume-valoare) și liste ordonate de valori (vectori).

Deși este posibil să se folosească funcția JavaScript eval() pentru a transforma o structură JSON într-un obiect JavaScript, acest lucru nu este recomandat din motive de securitate. În schimb, se recomandă utilizarea analizoarelor JSON scrise în JavaScript pentru o abordare mai sigură. Un alt avantaj al JSON în comparație cu XML este viteza de procesare. Datorită simplității sale, analizoarele JSON sunt mai ușor de construit și mai rapide. Viteza este crucială în aplicațiile AJAX, iar JSON se potrivește excelent pentru schimbul de date prin XMLHttpRequest, motiv pentru care acest format devine tot mai popular în dezvoltarea aplicațiilor Web 2.0. Dacă datele transmise de pe server sunt considerate sigure, se poate utiliza funcția eval() în partea de client pentru a crește viteza de procesare a datelor.

3. Implementare

În funcția de configurare (setup), se inițializează cele două module Bluetooth și Arduino, iar apoi se realizează conexiunea la rețeaua WiFi.

În funcția buclă (loop), se verifică dacă plăcuța (dispozitivul) s-a împerecheat cu un alt dispozitiv. Dacă împerecherea s-a realizat, se citește cererea (request-ul) transmisă de dispozitiv. Dacă cererea este de tipul {"action":"getData"}, înseamnă că trebuie să afișăm informațiile în următorul format: {id: int|string, name: string, image: string (url|base64)}.

Se realizează conexiunea la API folosind protocolul HTTP, iar datele sunt salvate într-un document JSON, care ulterior va fi deserializat și formatat pentru a obține șirul de caractere dorit.

Dacă cererea citită este de tipul {"action":"getDetails","id":"numarid"}, vom deserializa șirul de caractere pentru a extrage valoarea parametrului "numarid". Apoi vom interoga API-ul folosind expresia "id=numarid" și vom extrage datele într-un document JSON, care va fi deserializat și

formatat pentru a obține șirul de caractere în următorul format: {id: int|string, name: string, image: string (url|base64), description: string, teamId: string}.

4. Concluzie

Tehnologia reconfigurează modul în care oamenii învață, studiază, lucrează, comunică, gândesc și reflectă într-o lume tot mai conectată digital. Ea joacă un rol crucial în societatea contemporană, având atât efecte benefice, cât și negative asupra lumii și influențând viața de zi cu zi. Trăim într-o eră în care progresul tehnologic devine tot mai obișnuit, cu exemple notabile precum internetul și smartphone-urile. Inovația este fundamentală pentru progresul umanității, însă trebuie să fim conștienți de riscurile utilizării abuzive a tehnologiei, care poate afecta sănătatea mentală, capacitatea de concentrare și preocupările privind confidențialitatea datelor.

De ce este tehnologia atât de importantă în secolul 21? Tehnologia afectează aproape fiecare aspect al vieții în secolul 21, de la comunicare, securitate și eficiență în transport, până la accesul la alimente, incluziune financiară, asistență medicală, socializare și productivitate.

Iată cum tehnologia schimbă lumea de astăzi:

1. Conectivitate îmbunătățită prin dispozitivele Internet of Things (IoT): Petrecem tot mai mult timp online și conectivitatea ne oferă un nivel de confort inegalabil, însă aduce și vulnerabilități. Internetul obiectelor a revoluționat conectivitatea prin adăugarea de inteligență artificială, învățare automată și date inteligente. Din ce în ce mai multe dispozitive se conectează la internet, deschizând noi orizonturi pentru IoT.

2. Acces îmbunătățit la informații: Astăzi, pentru a afla ceva, avem nevoie doar de câteva clicuri pe telefon, computer sau prin intermediul asistenților inteligenți din casele noastre.

3. Comunicare digitală avansată prin social media și aplicații de mesagerie: Apelurile video și social media au devenit parte integrantă a vieții noastre, iar pandemia a accelerat această tendință. Companiile recunosc importanța managerilor de social media, care pot influența reputația unui brand.

4. Monitorizarea sănătății prin dispozitive inteligente: Dispozitivele de fitness au cunoscut o creștere semnificativă. Tehnologia ne permite să monitorizăm activitatea fizică într-un mod mai științific, oferind feedback instant și recomandări personalizate.

5. Munca flexibilă: Pandemia a popularizat munca de acasă, aducând cu sine o acceptare generală a acestei practici. Cu birourile închise, mulți oameni s-au conectat la muncă de la domiciliu.

6. Divertisment extins cu televizoare inteligente și platforme de streaming: Inovațiile tehnologice, cum ar fi jocurile video și televizoarele inteligente, ne oferă o gamă largă de opțiuni pentru a ne distra și a ne relaxa. Vizionarea filmelor nu se mai limitează la cinematografe, ci se desfășoară acum pe platforme digitale.

7. Noi oportunități în educație: Tehnologia deschide noi orizonturi în educație, cum ar fi învățarea la distanță, realitatea augmentată, realitatea virtuală și e-learning-ul. Acest lucru face educația mai accesibilă și permite oamenilor din întreaga lume să învețe de la instituții renumite, precum Harvard sau MIT, chiar de acasă, prin platformele educaționale online.

În ceea ce privește proiectul nostru din acest semestru, utilizând modulul ESP32, am reușit să implementăm un program care permite primirea și transmiterea de date (prin interogări HTTP) referitoare la preparate alimentare. Aceste date sunt apoi transmise către aplicația mobilă folosind Bluetooth Low Energy, deschizând noi posibilități pentru conectivitate și interacțiunea între dispozitive.

5. Bibliografie

1. https://www.bluetooth.com/

2. https://www.espressif.com/

3. https://arduinojson.org/

4. https://ro.wikipedia.org/

5. https://www.bbntimes.com/

6. Curs Servicii Internet

6. Cod

```
#include <Arduino.h>
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <ArduinoJson.h>
#include <WiFi.h>
#include <HTTPClient.h>


#define bleServerName "Proiect_PIA"
bool deviceConnected = false;

#define SERVICE_UUID "347d021a-d15c-4d15-8065-a0fcf77452f1"
#define CHARACTERISTIC_UUID "cbc94826-eee6-4334-acce-a3ca63239bf4"



BLECharacteristic characteristic(
  CHARACTERISTIC_UUID,
  BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE |
BLECharacteristic::PROPERTY_NOTIFY
```

```
);
BLEDescriptor *characteristicDescriptor = new
BLEDescriptor(BLEUUID((uint16_t)0x2902));

class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;
    Serial.println("Device connected");
  };
  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
    Serial.println("Device disconnected");
  }
};

class CharacteristicsCallbacks: public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *characteristic) {
    std::string data = characteristic->getValue();
    Serial.println(data.c_str()); // <-- This is the message sent from the
app, according to the specs

    DynamicJsonDocument doc(256);
    deserializeJson(doc, data);

    std::string action = doc["action"];
    std::string teamId = doc["teamId"];
    if (action == "getNetworks") {
      DynamicJsonDocument response(1024);

      // Code to get list of Wi-Fi networks in vicinity
      int numNetworks = WiFi.scanNetworks();

      // Add the Wi-Fi networks to the response
      for (int i = 0; i < numNetworks; i++) {
        JsonObject network = response.createNestedObject();
        network["ssid"] = WiFi.SSID(i);
        network["strength"] = WiFi.RSSI(i);
        network["encryption"] = WiFi.encryptionType(i);
        network["teamId"] = teamId;
        std::string jsonResponse;
        serializeJson(network, jsonResponse);
        characteristic->setValue(jsonResponse.c_str());
        characteristic->notify();
      }
    }
    const char* ssid = doc["ssid"];
    const char* password = doc["password"];
    if (action == "connect")
```

```
    {
        std::string ssid = doc["ssid"].as<std::string>();
        std::string password = doc["password"].as<std::string>();
        StaticJsonDocument<64> response;
        response["teamId"] = teamId;
        response["ssid"] = ssid;
        WiFi.begin(ssid.c_str(), password.c_str());
        int tries = 0;
        while (WiFi.status() != WL_CONNECTED && tries < 10) {
          delay(1000);
          tries++;
        }
        if (WiFi.status() == WL_CONNECTED) {
          response["connected"] = true;
        } else {
          response["connected"] = false;
        }
        std::string responseData;
        serializeJson(response, responseData);
        characteristic->setValue(responseData.c_str());
        characteristic->notify();
    }
     if( action == "getData")
    {

      {
      // Access the API and send back the data
      HTTPClient http;
      http.begin("http://proiectia.bogdanflorea.ro/api/breaking-
bad/characters");

      int httpCode = http.GET();
      if (httpCode == HTTP_CODE_OK) {
        String api_response = http.getString();
        // Parse the API response
        const size_t capacity = JSON_ARRAY_SIZE(17) + 17 * JSON_OBJECT_SIZE(4)
+ 500;
        DynamicJsonDocument api_doc(15000);

        // Deserialize the JSON std::string
        DeserializationError error = deserializeJson(api_doc, api_response);

        if (error) {
          Serial.print("deserializeJson() failed: ");
          Serial.println(error.c_str());
          return;
        }
```

```
        // Access each object in the JSON array
        JsonArray objects = api_doc.as<JsonArray>();
        for (JsonObject obj : objects) {
          // Extract the values from each object
          int id = obj["char_id"];
          const char* name = obj["name"];
          const char* imagePath = obj["img"];

          // Create a new object with the desired type
          JsonObject newObject = doc.to<JsonObject>();
          newObject["id"] = id;
          newObject["name"] = name;
          newObject["image"] = imagePath;
          newObject["teamId"] = "B22";  // Replace with your desired team ID

          // Convert the new object to a std::string
          std::string result;
          serializeJson(newObject, result);

          // Print the result
          characteristic->setValue(result.c_str());
          characteristic->notify();
        }
        http.end();
      }
    }
  }

  if (action == "getDetails") {
  String id = doc["id"];

  // Access the API and send back the data
  HTTPClient http;
  http.begin("http://proiectia.bogdanflorea.ro/api/breaking-
bad/character?char_id=" + id);

  int httpCode = http.GET();
  if (httpCode == HTTP_CODE_OK) {
    String api_response = http.getString();
    // Parse the API response
    const size_t capacity = JSON_ARRAY_SIZE(17) + 17 * JSON_OBJECT_SIZE(4) +
500;
    DynamicJsonDocument api_doc(15000);

    // Deserialize the JSON string
    DeserializationError error = deserializeJson(api_doc, api_response);

    if (error) {
```

```
      Serial.print("deserializeJson() failed: ");
      Serial.println(error.c_str());
      return;
    }

    // Access the JSON object
    JsonObject obj = api_doc.as<JsonObject>();

    // Extract the values from the JSON object
    int char_id = obj["char_id"];
    const char* name = obj["name"];
    const char* imagePath = obj["img"];
    const char* status = obj["status"];
    const char* birthday = obj["birthday"];
    const char* nickname = obj["nickname"];
    const char* portrayed = obj["portrayed"];

    // Create a new JSON object for the response
    DynamicJsonDocument responseDoc(JSON_OBJECT_SIZE(5) + JSON_ARRAY_SIZE(17)
+ 17 * JSON_OBJECT_SIZE(2) + capacity);
    JsonObject newObject = responseDoc.to<JsonObject>();

    // Assign values to the new object
    newObject["id"] = char_id;
    newObject["name"] = name;
    newObject["image"] = imagePath;
    newObject["teamId"]= "B22";

    const JsonArray& appearanceArray = obj["appearance"];
    String appearance;
    for (const JsonVariant& value : appearanceArray) {
      appearance += value.as<String>() + ", ";
    }
    appearance.remove(appearance.length() - 2, 2);

    const JsonArray& occupationArray = obj["occupation"];
    String occupation;
    for (const JsonVariant& value : occupationArray) {
      occupation += value.as<String>() + ", ";
    }
    occupation.remove(occupation.length() - 2, 2);

    // Format the description string
    char descriptionBuffer[500];
    sprintf(descriptionBuffer, "Birthday: %s.\nOccupation: %s.\nStatus:
%s.\nNickname: %s.\nPortrayed: %s.\nAppearance: %s.",
            birthday, occupation.c_str(), status, nickname, portrayed,
appearance.c_str());
```

```
    newObject["description"] = descriptionBuffer;

    // Convert the new object to a string
    String result;
    serializeJson(newObject, result);

    // Set the value and notify the characteristic
    characteristic->setValue(result.c_str());
    characteristic->notify();

    // Clean up
    http.end();
  }
}
  }
};

void setup() {
  Serial.begin(115200);
  BLEDevice::init(bleServerName);

  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
  BLEService *bleService = pServer->createService(SERVICE_UUID);
  bleService->addCharacteristic(&characteristic);
  characteristic.addDescriptor(characteristicDescriptor);
  characteristic.setCallbacks(new CharacteristicsCallbacks());
  bleService->start();
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pServer->getAdvertising()->start();
  Serial.println("Waiting a client connection to notify...");
}

void loop() {
  // Nothing to do here
}
```