



ISISS Marco Casagrande

IOT CASINO

FAGARAZ LUCA, VIEZZER TOMMASO, ZANCO SIMONE



INTERNET OF THINGS

Per **IoT** (Internet of Things) si intende l'estensione dell'Internet agli oggetti, i quali risultano in grado di comunicare senza la necessità dell'intervento umano.

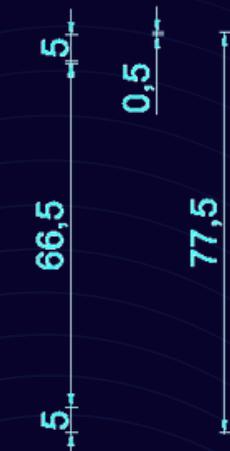
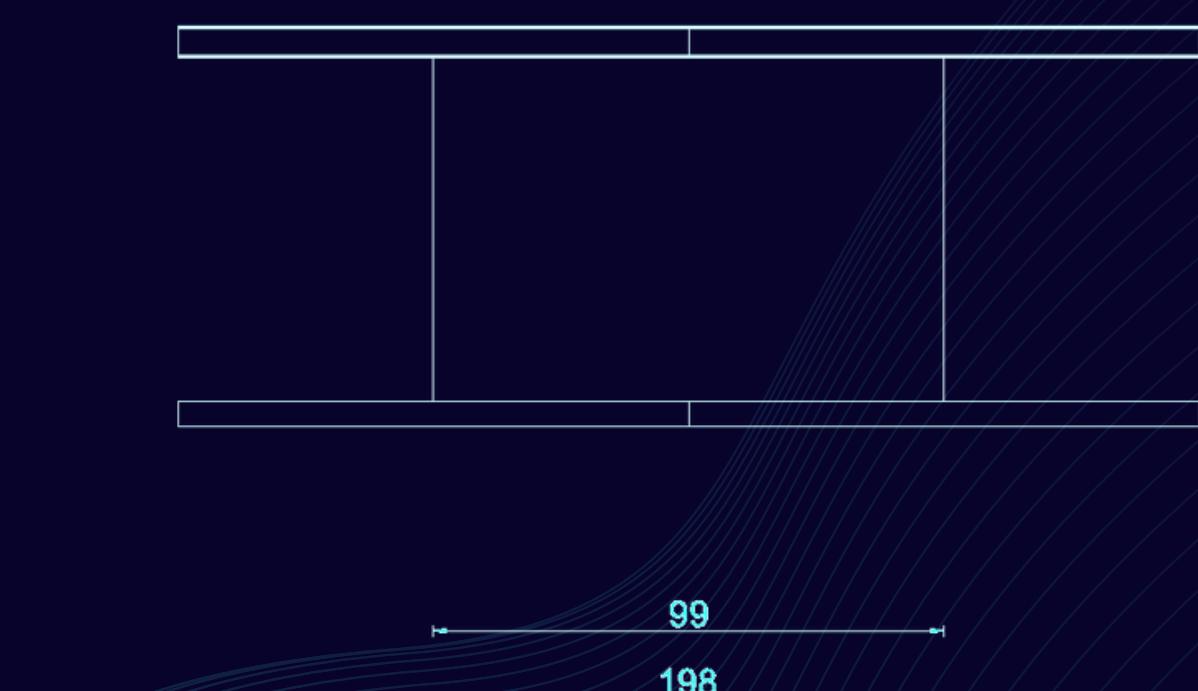
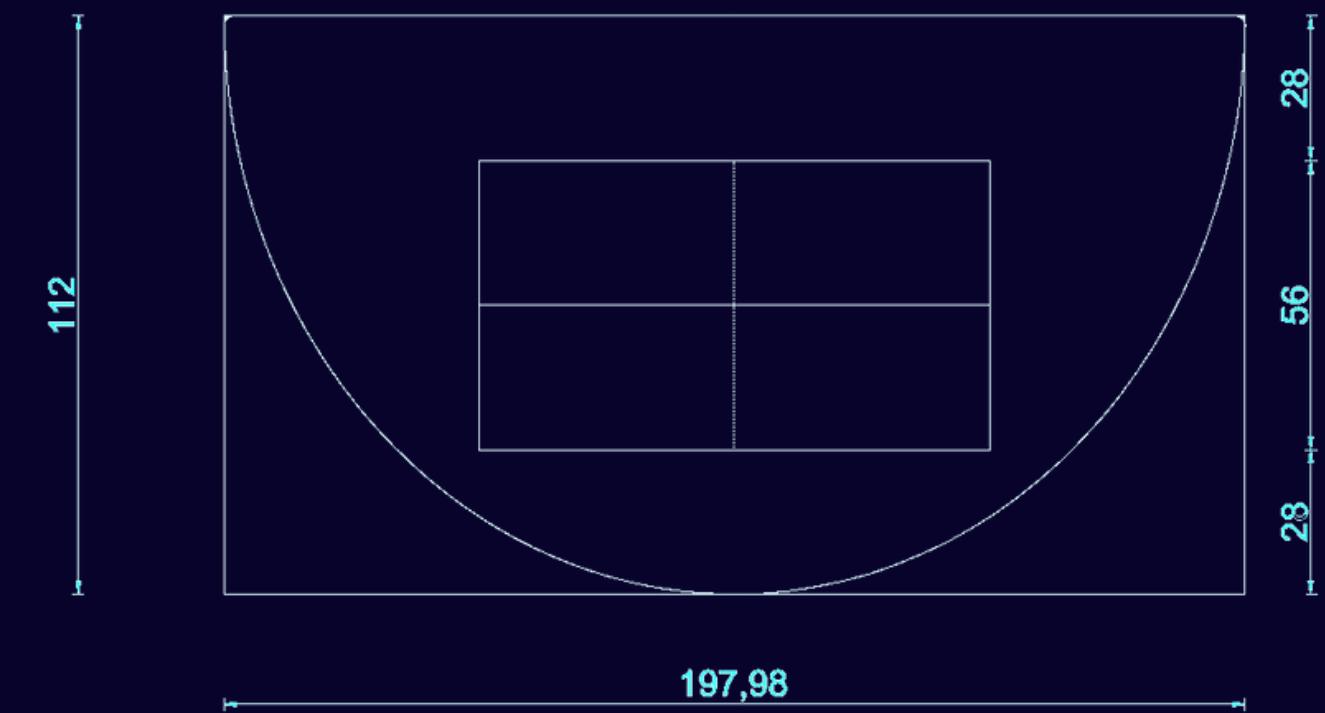


Internet of Things



IL TAVOLO: PROGETTO

- Larghezza: **120cm**
- Profondità: **40cm**
- Altezza: **6cm**

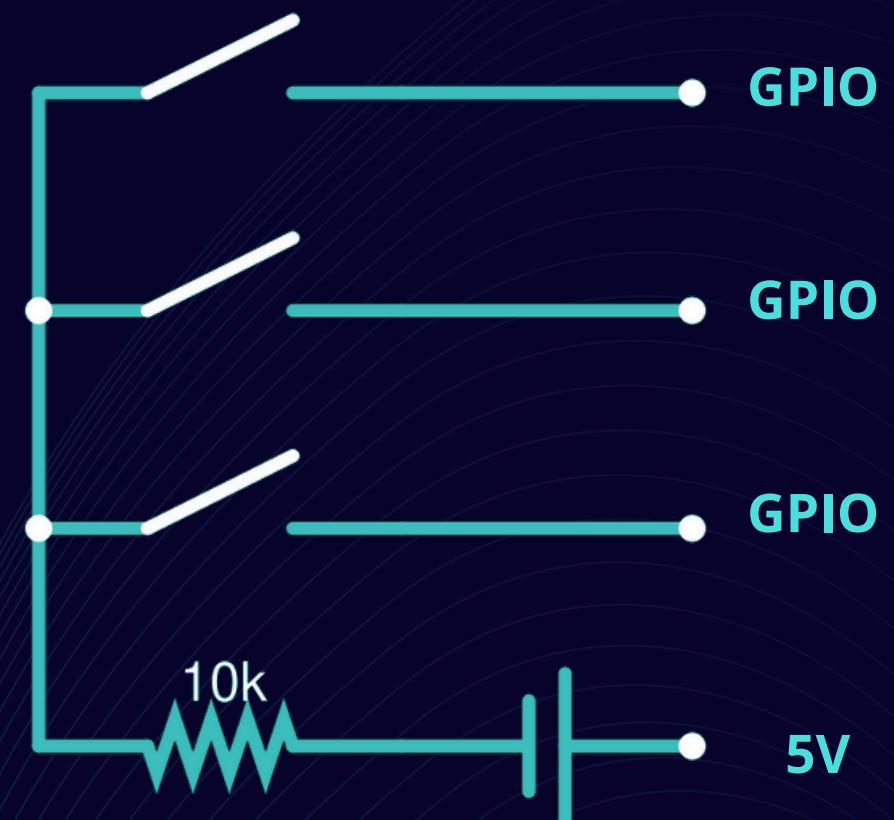


Primo progetto



IL TAVOLO: CIRCUITI

Le puntate sono gestite dal tavolo, la cui elettronica consiste di nove interruttori (tre per postazione) collegati a nove pin **GPIO** diversi della scheda ESP32 posta all'interno del tavolo.



Circuito di una postazione



IL TAVOLO: SOFTWARE

La scheda del tavolo è stata programmata in **C++** con le librerie di **Arduino**.

Ogni mano si sviluppa in quattro parti, mappate dalla variabile *mode*:

- **0:** Fase delle puntate;
- **1:** Turno dei giocatori;
- **2:** Turno del Dealer;
- **3:** Conteggio.

```
void manage_first_bet(int minusButton, int enterButton, int plusButton, int ID)
{
    static unsigned long t0 = 0;
    unsigned long t = millis(); // Timer

    bool minusState = digitalRead(minusButton);
    bool enterState = digitalRead(enterButton);
    bool plusState = digitalRead(plusButton);

    if (started)
    {
        if (plusState == 0 && lastRightStates[ID] == 1 && pots[ID] - step >= 0)
        {
            bets[ID] += step;
            pots[ID] -= step;
        }
        if (minusState == 0 && lastLeftStates[ID] == 1 && bets[ID] - step >= 0)
        {
            bets[ID] -= step;
            pots[ID] += step;
        }
        if (enterState == 0 && lastCenterStates[ID] == 1 && t - t0 > 5000)
        {
            positions_ready[ID] = true;
            return;
        }
        lastLeftStates[ID] = minusState;
        lastCenterStates[ID] = enterState;
        lastRightStates[ID] = plusState;
    }
    delay(1);
}
```

Gestione puntate



IL TAVOLO: SOFTWARE

La scheda del tavolo è stata programmata in **C++** con le librerie di **Arduino**.

Ogni mano si sviluppa in quattro parti, mappate dalla variabile *mode*:

- **0:** Fase delle puntate;
- **1:** Turno dei giocatori;
- **2:** Turno del Dealer;
- **3:** Conteggio.

```
void manage_game(int passButton, int doubleButton, int ID)
{
    if (turn == ID)
    {
        unsigned long t = millis(); // Timer
        static unsigned long t0 = t;

        bool passState = digitalRead(passButton);
        bool doubleState = digitalRead(doubleButton);

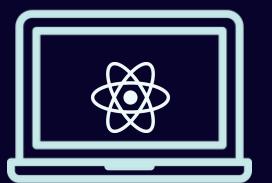
        if (passState == 0 && lastCenterStates[ID] == 1 && t - t0 > 2000)
        {
            turn++; // Pass
            return;
        }
        if (doubleState == 0 && lastLeftStates[ID] == 1 && t - t0 > 2000)
        {
            pots[ID] -= bets[ID];
            bets[ID] += bets[ID];
            turn++;
            return;
        }
    }
    delay(1);
}
```

Gestione partita



IL NETWORK

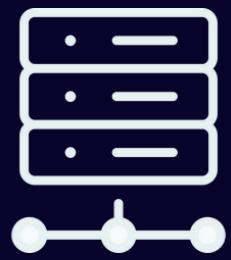
- Protocollo di trasporto e rete: **UDP/IP**
- Protocollo di applicazione: **HTTP**



192.168.4.*



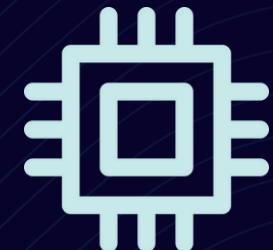
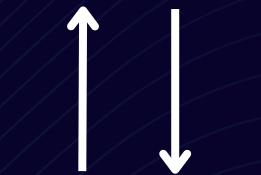
192.168.4.10



192.168.4.11



192.168.4.1



192.168.4.*



IL ROUTER

Il ruolo di router è svolto da una scheda **ESP32**.

L'indirizzo IP dell'AP (Access Point) è **192.168.4.1**, mentre gli indirizzi IP dei dispositivi collegati vengono assegnati automaticamente tramite DHCP (Dynamic Host Configuration Protocol) sulla Netmask **255.255.255.0**.





LA VIDEOCAMERA

La funzione di videocamera può essere svolta sia da una telecamera ESP32, sia da un cellulare Android correttamente configurato.

- Indirizzo IP (statico): **192.168.4.10**





LA VIDEOCAMERA: SOFTWARE

La telecamera ESP32 è stata programmata in **C++**, con le librerie di **Arduino**.

```
void serveJpg( )
{
    auto frame = esp32cam::capture();
    if (frame == nullptr)
    {
        Serial.println("CAPTURE FAIL");
        server.send(503, "", "");
        return;
    }

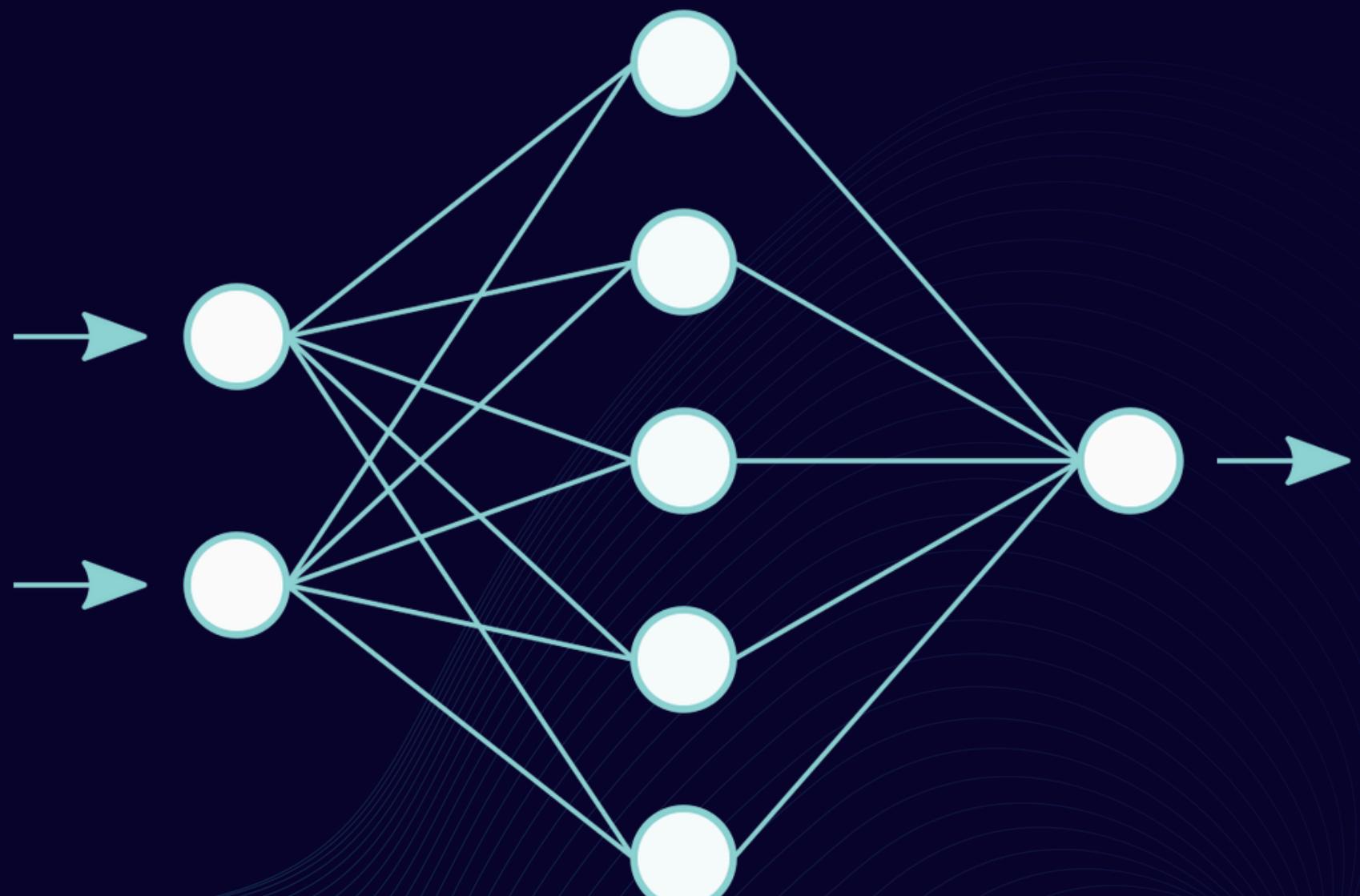
    server.setContentLength(frame->size());
    server.send(200, "image/jpeg");
    WiFiClient client = server.client();
    frame->writeTo(client);
}
```

LA RETE NEURALE

Per il riconoscimento delle carte è stata utilizzata una rete neurale.

$$X^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \dots \\ x_n^{(i)} \end{bmatrix}$$

Matrice degli input



Rete neurale



LA RETE NEURALE

Ad ogni collegamento tra due neuroni è assegnato un peso. Lo scopo è determinare questi pesi in modo che la rete riconosca in modo corretto le carte.

$$W^{(i)} = \begin{bmatrix} w_{1,1}^{(i)} & w_{1,2}^{(i)} & \dots & w_{1,n}^{(i)} \\ w_{2,1}^{(i)} & w_{2,2}^{(i)} & \dots & w_{2,n}^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1}^{(i)} & w_{k,2}^{(i)} & \dots & w_{k,n}^{(i)} \end{bmatrix}$$

Matrice dei pesi



LA RETE NEURALE

Definiamo la matrice di output di un layer come il risultato della funzione sigmoidea applicata alla somma tra una matrice di bias e il prodotto matriciale tra la matrice di input e la matrice dei pesi.

$$O^{(n)} = \sigma(W^{(n)}X^{(n)} + B^{(n)})$$

Matrice degli output

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Funzione sigmoidea



LA RETE NEURALE

Definiamo ora la funzione errore di un layer della rete neurale come metà della somma degli scarti quadratici medi degli output dall'output atteso. Il nostro scopo è determinare un minimo relativo della funzione definita come la somma di tutti gli errori su tutti i layer.

$$E(W^{(i)}) = \frac{1}{2} \sum_i (o_i - t_i)^2$$

Errore su un layer

$$E(W) = \frac{1}{2} \sum_j \sum_i (o_{ij} - t_{ij})^2$$

Errore su tutta la rete



LA RETE NEURALE

Per determinare un minimo possiamo utilizzare il metodo della **discesa del gradiente**.

$$\Delta w_{ij} = -R \frac{\partial E_d}{\partial w_{ij}}$$

Variazione di un peso



LA RETE NEURALE

Per determinare un minimo possiamo utilizzare il metodo della **discesa del gradiente**.

$$\Delta W = -R \nabla E$$

Variazione della matrice dei pesi



LA RETE NEURALE: DATASET

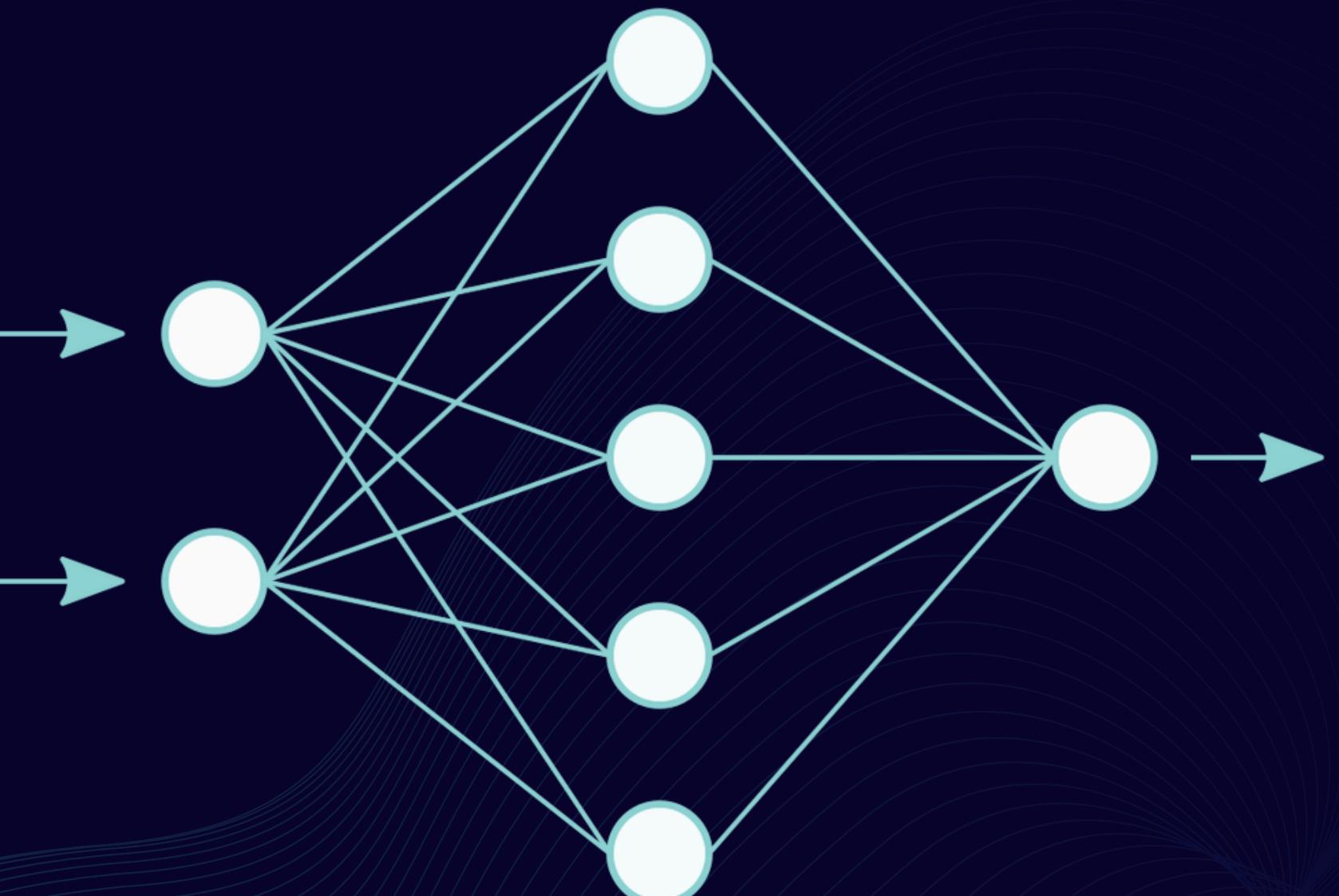
Per poter allenare la rete neurale è stato utilizzato un dataset generato tramite uno script in Python, il quale si occupa di generare immagini con carte casuali in posizioni casuali ed etichettarle.





LA RETE NEURALE: TRAINING

Per allenare la rete neurale, infine, è stato utilizzata la libreria di Python **YOLO**. Nel progetto finale è stato tuttavia utilizzato un altro modello di rete neurale già precedentemente allenato.





IL SERVER

La comunicazione tra i vari dispositivi è gestita interamente da un server centrale, programmato in JavaScript con le librerie **NodeJS** e **ExpressJS**.





IL SERVER: GESTIONE RICHIESTE

Le strutture dati e le variabili più importanti scambiati nel network sono:

- Il **Dealer** (dizionario);
- I **giocatori** (lista);
- La **modalità** (intero).

```
● ● ●  
  
let dealer = {  
  cards: [],  
  state: null,  
};  
  
let players = [  
  {pot: null, bet: 0, cards: [], position: "1", "state": null},  
  {pot: null, bet: 0, cards: [], position: "2", "state": null},  
  {pot: null, bet: 0, cards: [], position: "3", "state": null},  
]; // Players list  
  
let mode = null; // Part of the game
```



IL SERVER: GESTIONE RICHIESTE

Le richieste gestite dal server sono:

- **GET**: per ricevere dati dal server;
- **POST**: per inviare dati al server.

```
● ● ● ●  
app.post('/get-cards', (req, res) => {  
  const { data } = req.body;  
  if (!data) {  
    return res.status(400);  
  }  
  
  data.forEach(({ card, position }) => {  
    const isCardAlreadyPresent = dealer.cards.includes(card) || players.some(player =>  
      player.cards.includes(card));  
    if (!isCardAlreadyPresent) {  
      if (position === "Dealer") {  
        dealer.cards.push(card);  
      }  
      else {  
        const player = players.find(player => player.position === position);  
        if (player) {  
          player.cards.push(card);  
        }  
      }  
    }  
  });  
  return res.status(200).json({ players });  
}); // Receive and save the cards
```

Esempio di gestione richieste POST



IL SERVER: DEALER E BANKER

Oltre a comunicare con i dispositivi nella rete, il server gestisce lo scambio di informazioni tra due script scritti in **Python**, il **Dealer** e il **Banker**.





IL SERVER: DEALER

Il Dealer si attiva durante la modalità **1** (turno dei giocatori) e **2** (turno del Dealer) e si occupa di gestire i punteggi dei giocatori e del Dealer e determinare chi sballa, chi ottiene Black Jack, chi vince e chi perde.

```
def hand_value(player: object):

    s = 0
    numAces = 0

    for card in player["cards"]:
        if card[0] == 'A':
            s += 11
            numAces += 1
        elif card[0] == 'K' or card[0] == 'Q' or card[0] == 'J' or card[0] == '1':
            s += 10
        else:
            s += int(card[0])

    for ace in range(numAces):
        if s > 21:
            s -= 10

    return s
```

Calcolo punteggio di una mano



IL SERVER: DEALER

Il Dealer si attiva durante la modalità **1** (turno dei giocatori) e **2** (turno del Dealer) e si occupa di gestire i punteggi dei giocatori e del Dealer e determinare chi sballa, chi ottiene Black Jack, chi vince e chi perde.

```
● ● ●  
elif mode == 1:  
    if get_players_server():  
        for player in players:  
  
            if player["state"] == "playing":  
                if hand_value(player) == 21:  
                    player["state"] = "BJ"  
                if hand_value(player) > 21:  
                    player["state"] = "busted"  
  
            if send_states_server():  
                get_mode_server()  
                time.sleep(.5)
```

Turno dei giocatori



IL SERVER: DEALER

Il Dealer si attiva durante la modalità **1** (turno dei giocatori) e **2** (turno del Dealer) e si occupa di gestire i punteggi dei giocatori e del Dealer e determinare chi sballa, chi ottiene Black Jack, chi vince e chi perde.

```
elif mode == 2:
    if get_players_server():

        if hand_value(dealer) >= 17:

            if hand_value(dealer) > 21:
                dealer["state"] = "busted"
            if hand_value(dealer) == 21:
                dealer["state"] = "BJ"

        for player in players:

            if player["state"] == "playing":
                if hand_value(player) > hand_value(dealer):
                    player["state"] = "win"
                if hand_value(player) < hand_value(dealer):
                    player["state"] = "loss"
                if hand_value(player) == hand_value(dealer):
                    player["state"] = "push"

            if send_states_server():
                mode = 3
                send_mode_server() # Activate the Banker
                time.sleep(.5)
```

Turno del Dealer



IL SERVER: BANKER

Il Banker si attiva durante la modalità 3 (conteggio) e si occupa di calcolare e gestire la entrate e le uscite dei giocatori.

```
if mode == 3:  
    if get_players_server():  
        if dealer["state"] == "busted":  
            for player in players:  
                player["pot"] += 2 * player["bet"]  
        elif dealer["state"] == "BJ":  
            for player in players:  
                if player["state"] == "BJ":  
                    player["pot"] += player["bet"]  
        else:  
            for player in players:  
                if player["state"] == "win":  
                    player["pot"] += 2 * player["bet"]  
                if player["state"] == "push":  
                    player["pot"] += player["bet"]  
            for player in players:  
                player["bet"] = 0  
                player["state"] = "playing"  
            dealer["state"] = "playing"  
        if send_pots_server():  
            time.sleep(10)  
            send_reset_server()  
            mode = int(0)  
            send_mode_server()  
        else:  
            get_mode_server()  
            time.sleep(2)  
            time.sleep(.5)
```



IL SERVER: UI

Il server inoltre si occupa di rispondere ad ogni richiesta GET al ROOT con la **UI** (User Interface) di client o admin.



```
app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, clientUIPath));
}); // Sending default client UI
```

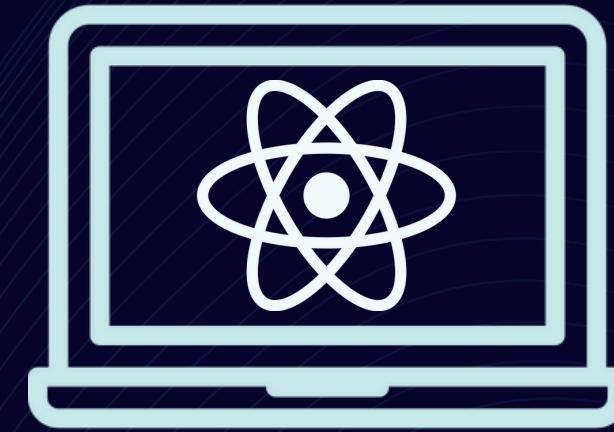


```
app.get('/admin', (req, res) => {
    const key = req.query.key;
    if (key === adminKey) {
        res.sendFile(path.join(__dirname, adminUIPath));
    } else {
        res.status(401).send('You are not allowed to access this resource');
    }
}); // Sending admin UI
```



L'APPLICAZIONE

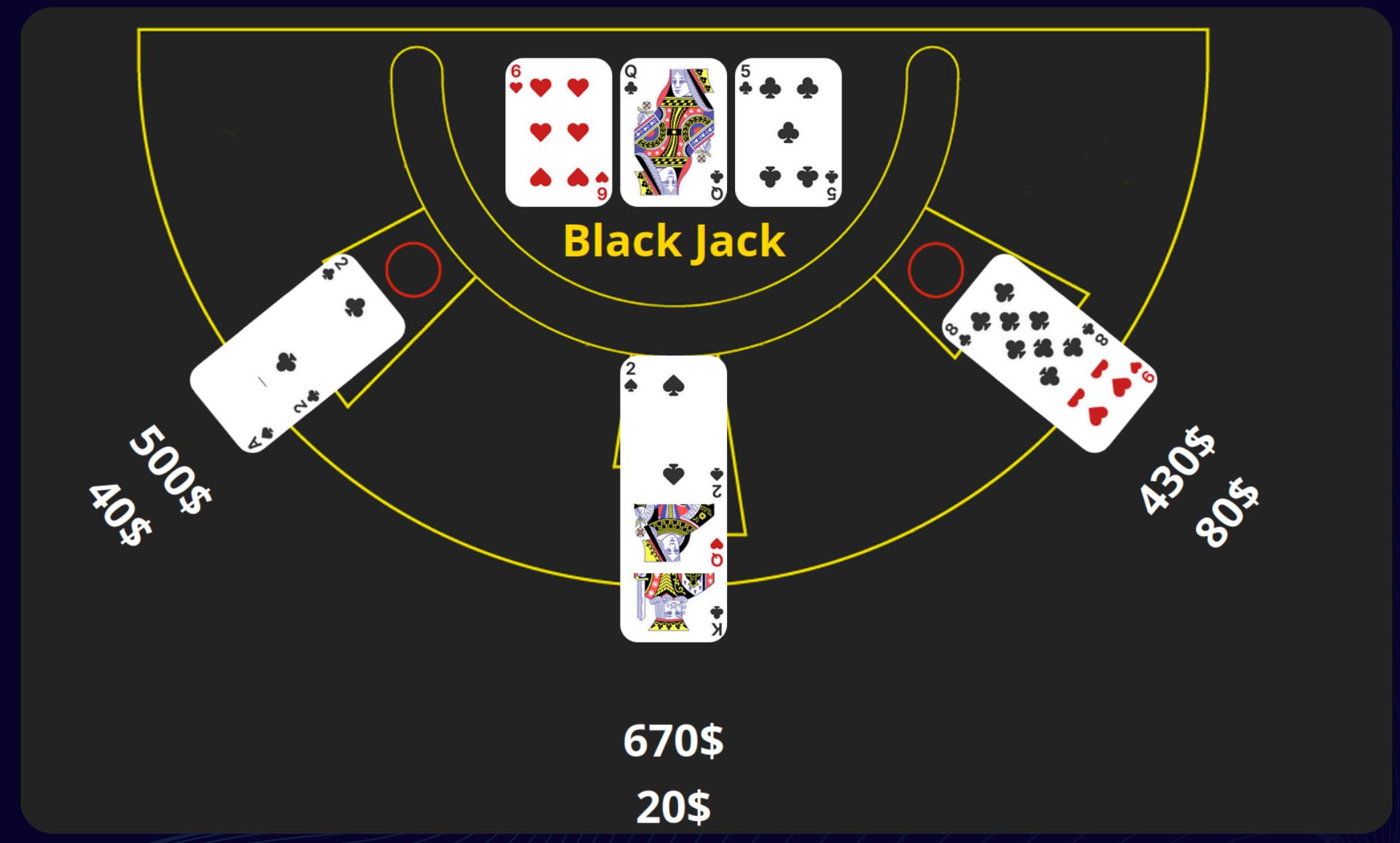
La Web App è stata sviluppata utilizzando **React**,
NPM (Node Package Manager) come gestore dei
pacchetti e **ViteJS** come ambiente di sviluppo.





L'APPLICAZIONE: CLIENT

Il lato client è servito ad ogni richiesta GET al percorso **ROOT** del server. Su questa pagina vengono aggiornati in tempo reale le carte del Dealer e dei giocatori e le rispettive puntate.





L'APPLICAZIONE: ADMIN

Il lato admin viene servito a richieste GET al percorso **/admin** aventi la chiave da amministratore e contiene informazioni statistiche sulla mano.

The screenshot shows a dark-themed dashboard titled "Dashboard" in large white font, flanked by a spade symbol on the left and a heart symbol on the right. Below the title, the text "Card Count:" is displayed in white, followed by the number "0" in yellow. At the bottom of the dashboard, there is a red footer bar containing the text "IoT Casino - Fagaraz Luca, Viezzer Tommaso, Zanco Simone - 2024" in white. The overall background of the slide features faint circular patterns.



GRAZIE PER L'ATTENZIONE

FAGARAZ LUCA, VIEZZER TOMMASO, ZANCO SIMONE