

# Ping-Povo

DELIVERABLE 4 - G21  
Marco Strada e Matej Del Coco

Dicembre 9, 2023

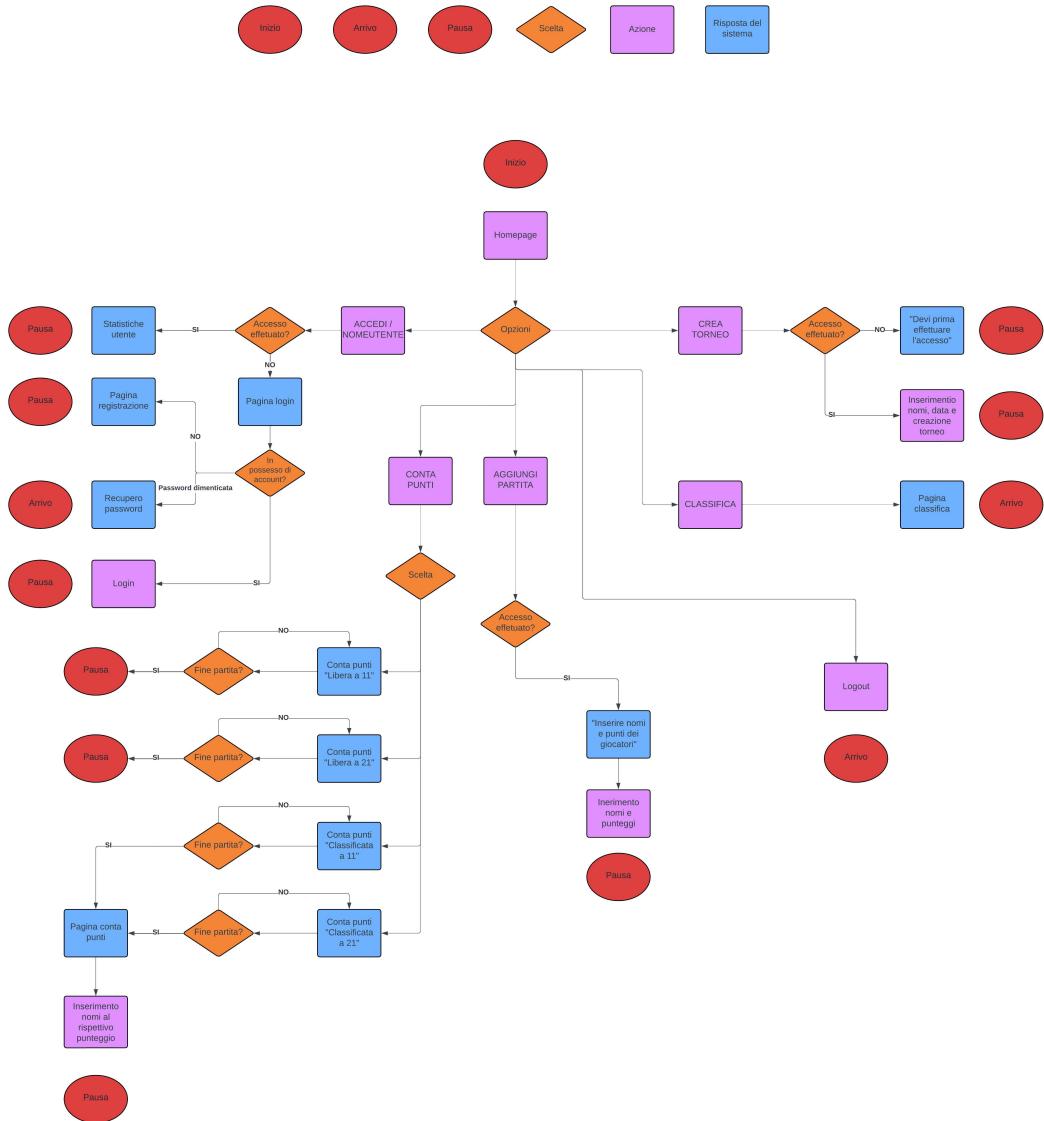
## Contents

<b>1 User flow</b>	<b>3</b>
1.1 Implementazione e Documentazione dell'Applicazione . . . . .	4
1.2 Project Structure . . . . .	4
1.3 Project Dependencies . . . . .	6
1.4 Database . . . . .	6
1.4.1 Database utenti . . . . .	7
1.4.2 Database amicizie . . . . .	7
1.4.3 Database tornei . . . . .	7
<b>2 Documentazione API</b>	<b>8</b>
2.1 Resources Extractions . . . . .	8
2.2 Resources Models . . . . .	9
2.2.1 Documentazione Swagger . . . . .	10
2.3 Implementazione API . . . . .	11
2.3.1 Creazione amicizia . . . . .	11
2.3.2 Visualizzazione amicizia . . . . .	12
2.3.3 Creazione profilo . . . . .	12
2.3.4 Login . . . . .	13
2.3.5 Visualizzazione profilo . . . . .	13
2.3.6 PP calculator . . . . .	14
2.3.7 Visualizza utenti . . . . .	15
2.3.8 Creazione torneo . . . . .	15
2.3.9 Verifica esistenza torneo . . . . .	16
2.3.10 ICS torneo . . . . .	16
<b>3 Implementazione frontend</b>	<b>17</b>
3.0.1 Home page . . . . .	17
3.0.2 Login . . . . .	18
3.0.3 Registrazione . . . . .	18
3.0.4 Recupero password . . . . .	18
3.0.5 Logout . . . . .	19
3.1 Profilo utente . . . . .	19
3.2 Lista amici . . . . .	19
3.2.1 Scegli modalità . . . . .	20
3.2.2 Conta punti . . . . .	20
3.2.3 Aggiungi partita . . . . .	21
3.3 Classifica . . . . .	21
3.3.1 Crea torneo . . . . .	22
<b>4 Esecuzione del codice</b>	<b>23</b>
4.1 Esecuzione in locale . . . . .	23

<b>5 Testing</b>	<b>24</b>
5.1 Risultato test login . . . . .	24

## 1 User flow

In questa sezione del documento vengono illustrate le diverse possibilità di interazione con l'applicativo con il prototipo che abbiamo sviluppato. Attraverso un diagramma di flusso, abbiamo delineato il percorso che un utente autenticato, o non, può seguire durante l'utilizzo di Ping-Povo.



## 1.1 Implementazione e Documentazione dell'Applicazione

Nelle fasi precedenti abbiamo delineato le diverse funzionalità integrate nella nostra piattaforma e abbiamo illustrato come gli utenti finali potranno beneficiarne. Il sistema è stato realizzato sfruttando Node.js per il lato server, mentre per il frontend abbiamo adottato SvelteKit, un framework che sfrutta TypeScript, CSS ed HTML. Per la gestione dei dati, ci siamo affidati a Supabase.

## 1.2 Project Structure

Qui sotto è riportata la struttura del back-end, gestita in modo semplice e intuitivo.

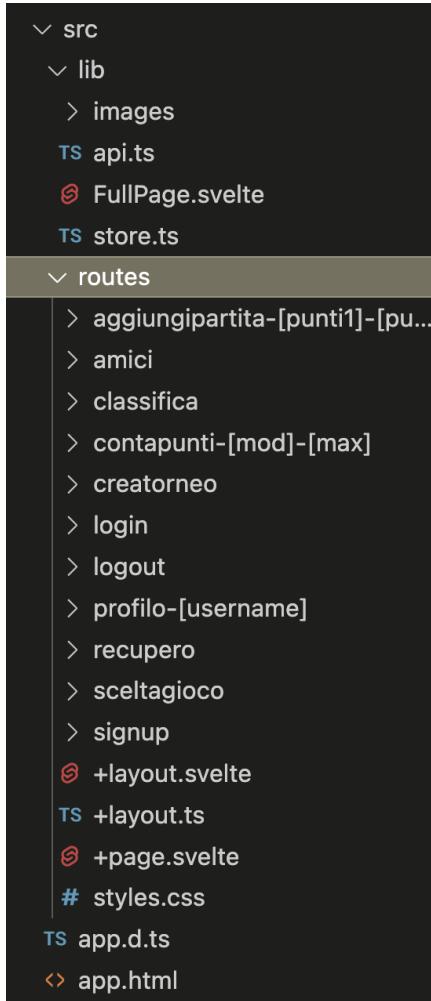
```
✓ dist
  JS api.js
  JS index.js
  JS supabase.js
> node_modules
✓ source
  TS api.ts
  TS index.ts
  TS supabase.ts
  .env
  .gitignore
  {} package-lock.json
  {} package.json
  README.md
  {} swagger.json
  tsconfig.json
```

All'interno di **api.ts** sono contenute le API del back-end. Queste verranno analizzate in seguito una ad una.

**Index.ts** gestisce le rotte API e fornisce documentazione Swagger.

**Supabase.ts** definisce modelli di dati per l'uso di Supabase, stabilendo i tipi per tabelle, operazioni di inserimento e aggiornamento, nonché enum nel database.

La figura seguente raffigura la struttura del front-end. Le varie cartelle contengono al loro interno le pagine della relativa scena nel front-end, come previsto da Sveltekit. Queste sono le stesse seguite dallo user-flow descritto in precedenza.



**Aggiungipartita:** riassegnazione dei *PP* in base al risultato di una partita.

**Amici:** permette di vedere la lista dei propri amici.

**Classifica:** permette di visualizzare la classifica della stagione attuale.

**Contapunti:** interfaccia per aiutare l'assegnazione dei punti in una partita.

**Creatorneo:** procedere alla creazione del torneo inserendo i nomi dei partecipanti e la data.

**Login:** login.

**Logout:** logout.

**Profilo:** visualizzazione delle statistiche del giocatore.

**Recupero:** recupero delle credenziali.

**Sceltagioco:** selezionare la modalità di gioco.

**Signup:** creazione del proprio profilo.

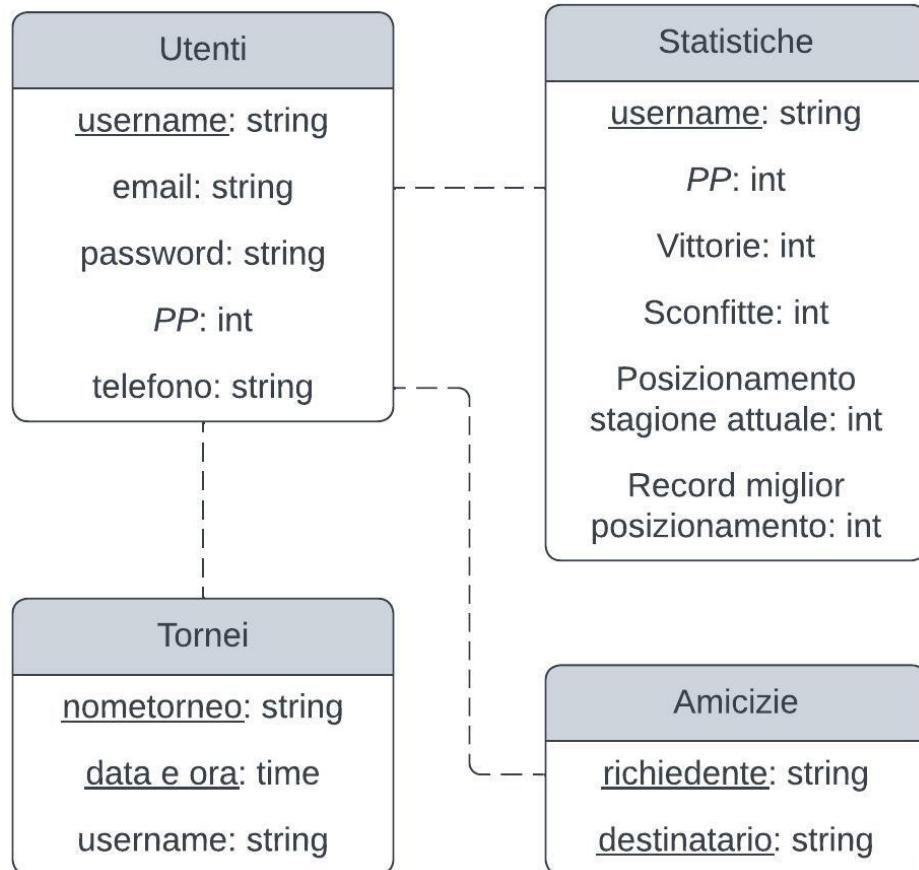
### 1.3 Project Dependencies

I seguenti moduli Node sono stati utilizzati e aggiunti al file Package.Json:

- "@supabase/supabase-js": "^2.26.0"
- "@types/express": "^4.17.17"
- "@types/node": "^20.4.1"
- "dotenv": "^16.1.4"
- "swagger-ui-express": "^5.0.0"
- "typescript": "^5.3.3"

### 1.4 Database

Per la gestione dei dati necessari al funzionamento della piattaforma, abbiamo sviluppato uno schema SQL che rappresenta la struttura delle tabelle. Con questo approccio semplice e chiaro, siamo riusciti a gestire tutti i dati necessari per lo sviluppo della web-app.



### 1.4.1 Database utenti

Visualizzazione degli utenti attualmente registrati sul database:

username	PP	email	telefono
G21	2100	G21@unitn	2121212121
user6	1500	user6@user	1234567890
matejdelcoco	1366	matejdelcoco@gmail.com	0987654321
user2	1300	user2@user	1234567809
user4	1000	user4@user	1234567890
user5	1000	user5@user	1234567890
user1	1000	user1@user	1234567899
user3	932	user3@user	0987654322
marcostrada	827	marcostrada@gmail.com	1234567890

### 1.4.2 Database amicizie

Visualizzazione di alcune amicizie presenti sul database:

utente_richiedente	utente_destinatario
marcostrada	matejdelcoco
user2	user3
user1	user3
user3	matejdelcoco
G21	marcostrada
G21	matejdelcoco
G21	user6

### 1.4.3 Database tornei

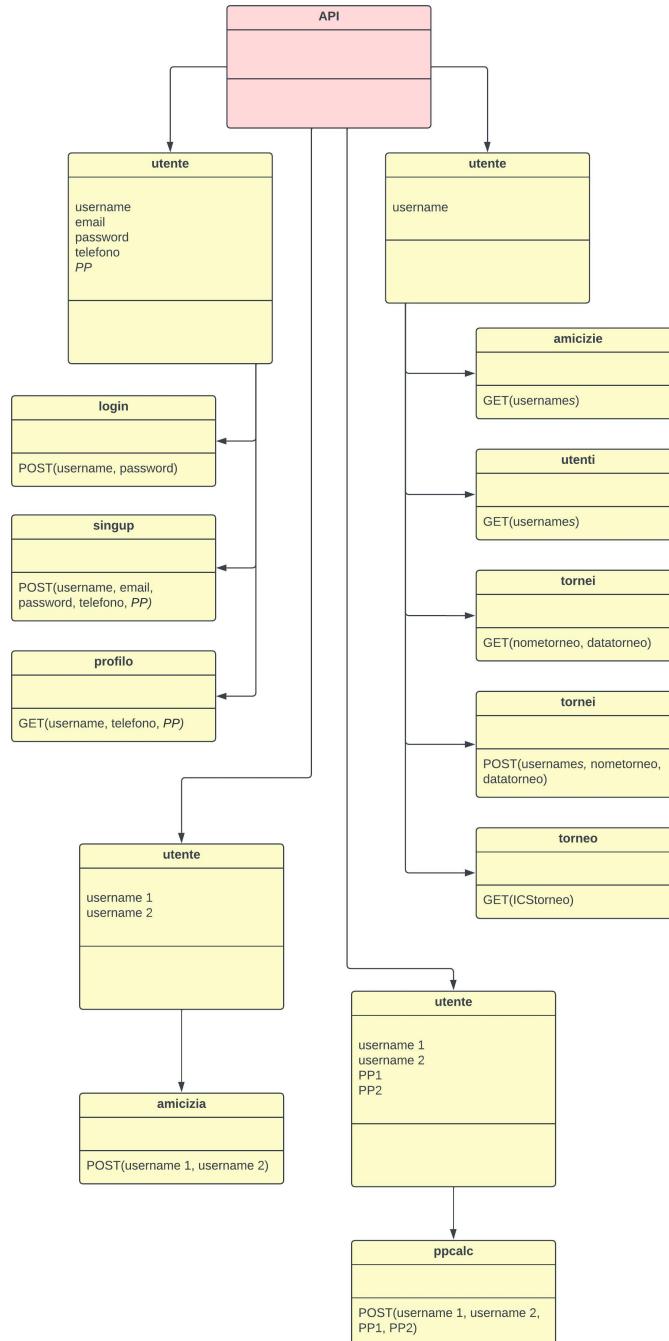
Visualizzazione dei tornei creati sul database:

username	nometorneo	d...
marcostrada	D4 CHAMPIONSHIP	2023-11-22 11:30:00+00
G21	D4 CHAMPIONSHIP	2023-11-22 11:30:00+00
matejdelcoco	D4 CHAMPIONSHIP	2023-11-22 11:30:00+00
user1	D4 CHAMPIONSHIP	2023-11-22 11:30:00+00
user2	D4 CHAMPIONSHIP	2023-11-22 11:30:00+00
user3	D4 CHAMPIONSHIP	2023-11-22 11:30:00+00
user4	D4 CHAMPIONSHIP	2023-11-22 11:30:00+00
user5	D4 CHAMPIONSHIP	2023-11-22 11:30:00+00

## 2 Documentazione API

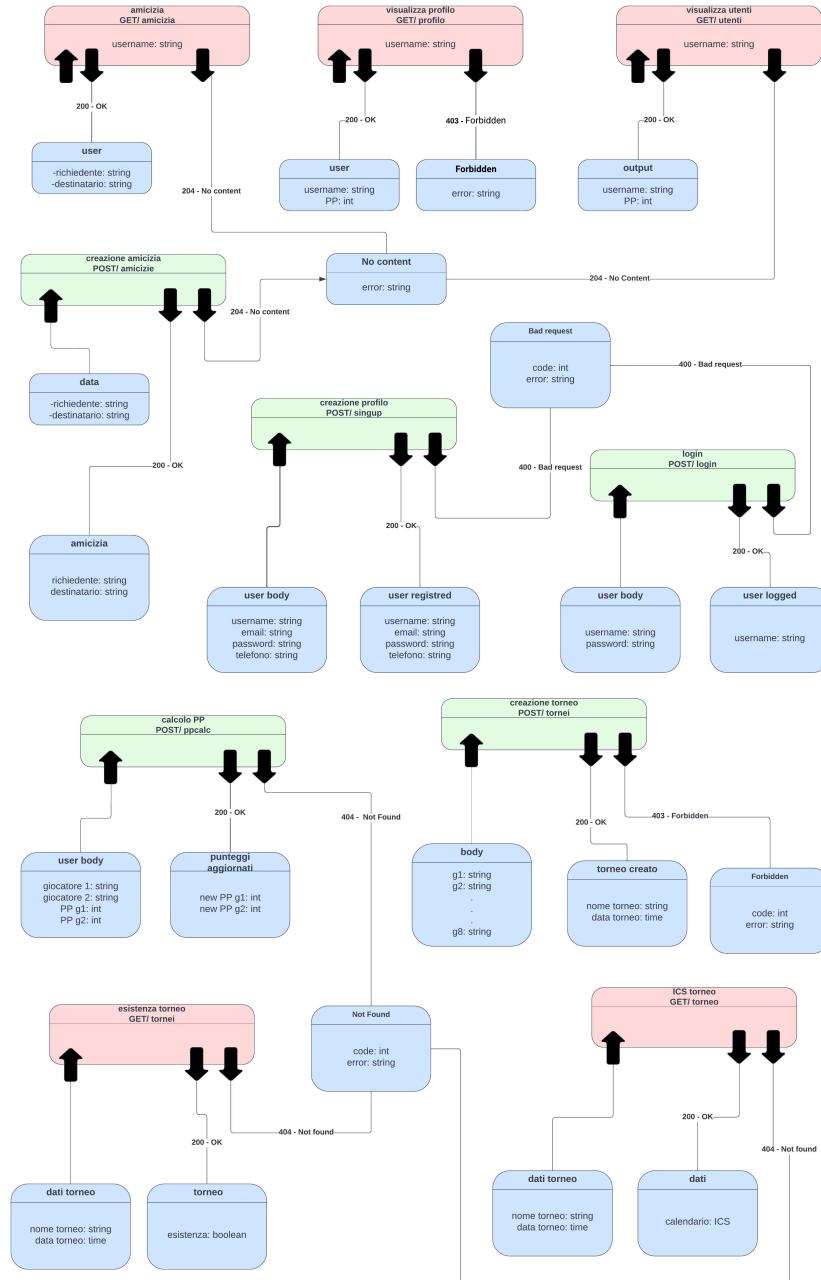
### 2.1 Resources Extractions

Nel seguente diagramma vengono rappresentate le classi e le loro relazioni con le API del progetto.



## 2.2 Resources Models

In questa sezione vengono rappresentati i diagrammi **Resources Models**, uno per ogni risorsa. In ogni API resource è presente l'URI per accedere al servizio rappresentato. Per una visione più chiare le **API/GET** sono colorate di rosso, mentre le **API/POST** di verde.



### 2.2.1 Documentazione Swagger

Qui di seguito proposta anche la documentazione eseguita con **Swagger**, con i relativi modelli.

<p><b>POST</b> /amicizie Invia una richiesta di amicizia</p> <p><b>GET</b> /amicizie Ottieni elenco amicizie utente</p> <p><b>POST</b> /signup Registrazione utente</p> <p><b>POST</b> /Login Login utente</p> <p><b>GET</b> /profilo Ottieni il profilo utente</p> <p><b>POST</b> /ppcalc Calcola il punteggio PP</p> <p><b>GET</b> /utenti Ottieni elenco utenti</p> <p><b>GET</b> /tornei Ottieni elenco tornei</p> <p><b>POST</b> /tornei Crea un nuovo torneo</p> <p><b>GET</b> /torneo Ottieni dettagli torneo</p>	
<p><b>Utente</b> ↴ {</p> <ul style="list-style-type: none"> <li>PP integer Punteggio PP dell'utente</li> <li>username string Nome utente</li> </ul> <p>}</p>	<p><b>Amicizia</b> ↴ {</p> <ul style="list-style-type: none"> <li>username string Nome utente amico</li> <li>numero string Numero di telefono dell'amico</li> <li>PP integer Punteggio PP dell'amico</li> </ul> <p>}</p>
<p><b>Torneo</b> ↴ {</p> <ul style="list-style-type: none"> <li>username string Nome utente creatore del torneo</li> <li>nometorneo string Nome del torneo</li> <li>date string Data e ora del torneo</li> </ul> <p>}</p>	<p><b>DettagliTorneo</b> ↴ {</p> <ul style="list-style-type: none"> <li>start ↴ [ Data di inizio torneo integer ]</li> <li>duration ↴ { description: Durata del torneo hours integer Durata in ore }</li> <li>title string Nome del torneo</li> <li>description string Descrizione del torneo</li> <li>location string Luogo del torneo</li> <li>status string Stato del torneo</li> </ul> <p>}</p>

## 2.3 Implementazione API

Qui di seguito saranno indicate le foto del codice delle singole API e la loro intestazione.

### 2.3.1 Creazione amicizia

API per creare un'amicizia tra due utenti.

```
app.post("/api/amicizie", async (req, res) => {
  const { username, amico } = req.body;
  //check for forbidden
  const { data, error } = await supabaseClient
    .from("utenti")
    .select("*")
    .eq("username", amico);
  if (error)
    console.error(error);
  if (data?.length == 0) {
    res.status(404).send(`Non ho trovato utenti con nome ${amico}`);
    return;
  }
  else if (data) {
    const { data: updatedRows, error } = await supabaseClient
      .from("amicizie")
      .insert({ utente_destinatario: username, utente_richiedente: amico })
      .select().single();
    if (error) {
      if (error.code == "23505") {
        res.status(204);
        return;
      }
    }
    if (updatedRows) {
      res.status(200).send("OK");
      return;
    }
  }
});
```

### 2.3.2 Visualizzazione amicizia

API per ottenere la lista delle amicizie.

```
app.get("/api/amicizie", async (req, res) => {
  const utente = req.query.utente;
  const { data, error } = await supabaseClient
    .from("amicizie")
    .select("*", { utente: utente_destinatario(*) })
    .or(`utente_richiedente.eq.${utente},utente_destinatario.eq.${utente}`);
  if (error) {
    console.log(error);
    throw error;
  }
  if (data) {
    if (data.length == 0) {
      res.status(204).send();
      return;
    }
    res.status(200).send(data.map(e => {
      console.log(e.utenti);
      return {
        username: e.utente_destinatario,
        // @ts-ignore
        numero: e.utenti.telefono,
        // @ts-ignore
        PP: e.utenti.PP
      };
    }));
    return;
  }
});
```

### 2.3.3 Creazione profilo

API per creare un profilo utente.

```
app.post("/api/signup", async (req, res) => {
  const { email, password, telefono, username, } = req.body;
  //check for forbidden
  const { data, error } = await supabaseClient
    .from("utenti")
    .insert({
      email,
      password,
      telefono,
      username,
    })
    .select("*");
  if (data) {
    res.status(200).send("Utente registrato con successo");
  }
  else {
    console.log(error);
    res.status(400).send("Impossibile loggare");
  }
});
```

### 2.3.4 Login

API per il login con nickname e password.

```
app.post("/api/login", async (req, res) => {
  const { username, password } = req.body;
  //check for forbidden
  const { data, error } = await supabaseClient
    .from("utenti")
    .select("*")
    .eq("username", username)
    .eq("password", password)
    .single();
  console.log(req.body, username, password, data);
  if (data) {
    res.status(200).send("Utente loggato con successo");
  }
  else {
    res.status(403).send("Impossibile loggare");
  }
});
```

### 2.3.5 Visualizzazione profilo

API per ottenere le statistiche di un profilo.

```
app.get("/api/profilo", async (req, res) => {
  const { username } = req.query;
  console.log(username);
  const { data, error } = await supabaseClient
    .from("utenti")
    .select("*")
    .eq("username", username)
    .single();
  console.log(req.body, username);
  if (data) {
    res.status(200).send(data);
  }
  else {
    res.status(403).send("Impossibile trovare");
  }
});
```

### 2.3.6 PP calculator

API per il ricalcolo dei *PP* dopo una partita classificata vinta o persa.

```
// //PP calculator
app.post("/api/ppcalc", async (req, res) => {
  let { winner, loser } = req.body;
  console.log(winner, loser);
  const winner = winner_;
  const loser = loser_;
  //winner PASSATO VINCE SEMPRE
  if (!winner || !loser) {
    res.status(400).send("Input sbagliato");
    return;
  }
  try {
    const { data, error } = await supabaseClient
      .from("utenti")
      .select("*")
      .or(`username.eq.${winner},username.eq.${loser}`);
    let elo1 = -1;
    let elo2 = -1;
    if (error)
      throw error;
    if (data) {
      data.map((x) => {
        if (x.username == winner) {
          elo1 = x.PP;
        }
        if (x.username == loser) {
          elo2 = x.PP;
        }
      });
      if (elo1 == -1 || elo2 == -1) {
        return res.status(404).send("Uno dei due utenti non esiste o ha eliminato l'account durante la partita");
      }
      let pa = 1 / (1 + 10 * ((elo1 - elo2) / 400));
      let pb = 1 / (1 + 10 * ((elo2 - elo1) / 400));
      pa = 30 * (1 - pa) + 10;
      pb = -(30 * (1 - pb)) + 10;
      elo1 = elo1 + pa;
      elo2 = elo2 + pb;
      elo1 = Math.round(elo1);
      elo2 = Math.round(elo2);
      console.log(elo1, elo2);
    }
    const { data: updatedRows, error: error2 } = await supabaseClient
      .from("utenti")
      .update({ username: loser, PP: elo2 })
      .eq("username", loser)
      .select();
    const { data: updatedRows3, error: error3 } = await supabaseClient
      .from("utenti")
      .update({ username: winner, PP: elo1 })
      .eq("username", winner)
      .select();
    if (error2 || error3) {
      throw error2 || error3;
    }
    if (updatedRows && updatedRows3) {
      res.status(200).send();
      return;
    }
  }
})
```

### 2.3.7 Visualizza utenti

API per ottenere i dati di tutti gli utenti con il fine di creare la classifica.

```
app.get("/api/utenti", async (req, res) => {
  const { data, error } = await supabaseClient
    .from("utenti")
    .select("PP,username")
    .order("PP", { ascending: false });
  if (error)
    throw error;
  if (data) {
    res.status(200).send(data);
    return;
  }
  res.status(204).send();
});
```

### 2.3.8 Creazione torneo

API per creare un torneo.

```
app.post("/api/tornei", async (req, res) => {
  const { UTENTI, nometorneo, orario } = req.body;
  const ora = new Date(orario);
  let ora2;
  const { data, error } = await supabaseClient
    .from("tornei")
    .select("*");
  if (error)
    throw error;
  data.map(x => {
    if (x.nometorneo == nometorneo) {
      res.status(403).send("Tournament name already exists");
      return;
    }
    ora2 = new Date(x.date);
    if (ora2.getTime() - ora.getTime() <= 1000 * 60 * 120 &&
        ora2.getTime() - ora.getTime() > 0) {
      return res.status(403).send("Orario non disponibile a Povo!");
    }
  });
  UTENTI.map(async x => {
    console.log(x);
    const { data: updatedRows, error } = await supabaseClient
      .from("tornei")
      .insert({ username: x, nometorneo: nometorneo, date: orario })
      .select();
    if (error)
      console.error(error);
  });
  return res.status(200).send();
});
```

### 2.3.9 Verifica esistenza torneo

API per verificare l'esistenza di un torneo, in modo tale da non consentirne la sovrapposizione.

```
app.get("/api/tornei", async (req, res) => {
  const { utente } = req.query;
  if (!utente) {
    res.status(502).send();
    return;
  }
  const { data, error } = await supabaseClient
    .from("tornei")
    .select("*")
    .eq("username", utente);
  if (error)
    throw error;
  if (data) {
    if (data.length == 0) {
      return res.status(204).send();
    }
    return res.status(200).send(data);
  }
  return res.status(404).send();
});
```

### 2.3.10 ICS torneo

API per ottenere il file .ics del torneo.

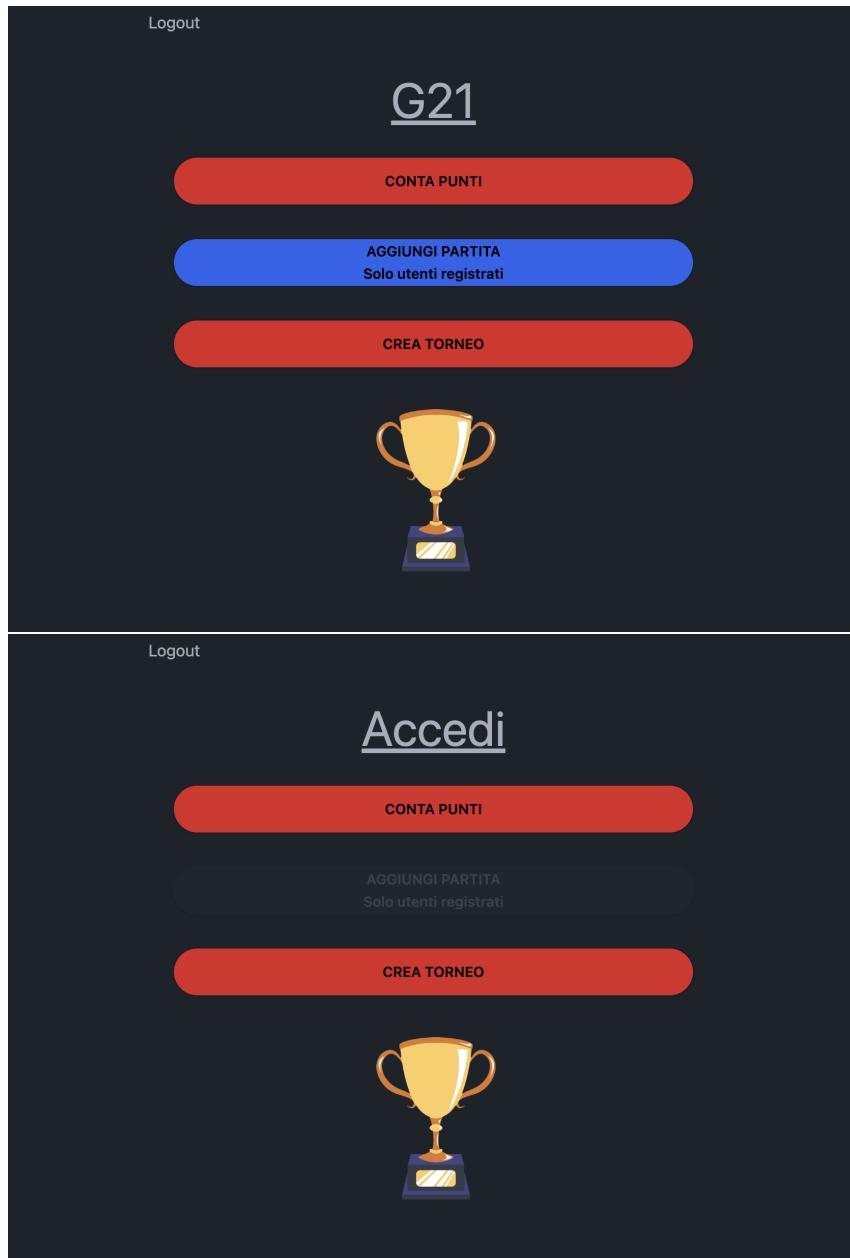
```
//API per prendere l'ICS
app.get("/api/torneo", async (req, res) => {
  const { torneo } = req.query;
  const { data, error } = await supabaseClient
    .from("tornei")
    .select("*")
    .eq("nometorneo", torneo);
  if (error)
    throw error;
  if (data) {
    if (data.length == 0) {
      return res.status(404).send("Torneo non trovato");
    }
    let ora = new Date(data[0].date);
    const event = {
      start: [ora.getFullYear(), ora.getMonth(), ora.getDate(), ora.getMinutes(), ora.getSeconds()],
      duration: { hours: 2 },
      title: torneo,
      description: 'Il classico torneo di Povo ritorna!',
      location: '38123 Trento, Autonomous Province of Trento',
      status: 'CONFIRMED'
    };
    return res.status(200).send(event);
  }
});
```

### 3 Implementazione frontend

In questa sezione del documento mostriamo le schermate raggiungibili dal nostro front-end.

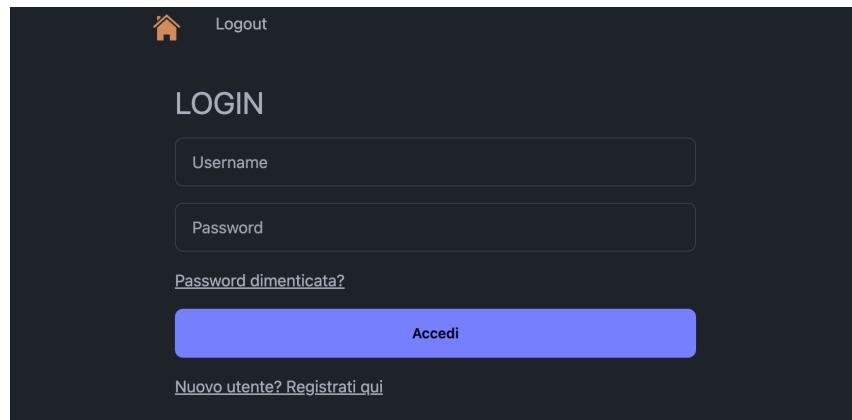
#### 3.0.1 Home page

Schermata homepage. Nell'immagine superiore si vede un utente loggato con nome utente: "G21", che ha la possibilità di usare la funzione aggiungi partita. Nell'immagine sottostante invece l'utente non è loggato e non ha a disposizione tutte le funzioni.



### 3.0.2 Login

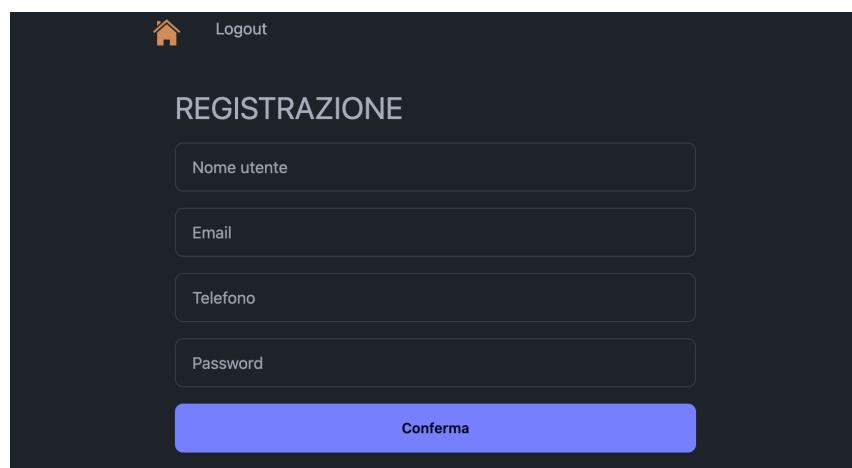
Schermata di login con la possibilità di accedere alla schermata di recupero password.



The screenshot shows a dark-themed login interface. At the top left is a house icon and the word "Logout". In the center, the word "LOGIN" is displayed in large capital letters. Below it are two input fields: one labeled "Username" and another labeled "Password". Underneath these fields is a link "Password dimenticata?". A large blue button at the bottom contains the text "Accedi". At the very bottom, there is a link "Nuovo utente? Registrati qui".

### 3.0.3 Registrazione

Schermata per la registrazione.



The screenshot shows a dark-themed registration interface. At the top left is a house icon and the word "Logout". In the center, the word "REGISTRAZIONE" is displayed in large capital letters. Below it are four input fields: "Nome utente", "Email", "Telefono", and "Password". A large blue button at the bottom contains the text "Conferma".

### 3.0.4 Recupero password

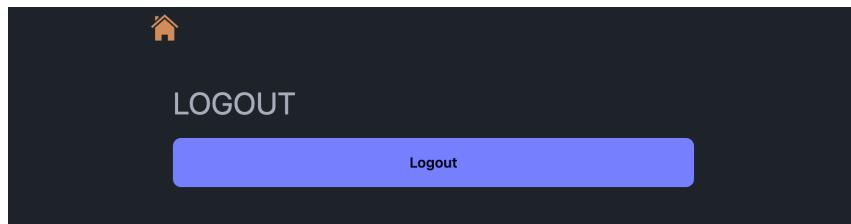
Schermata per il recupero della password.



The screenshot shows a dark-themed password recovery interface. At the top left is a house icon and the word "Logout". In the center, the words "RECUPERO CREDENZIALI" are displayed in large capital letters. Below it is a single input field labeled "Email". Underneath the input field is a placeholder text "Inserisci la tua email per ricevere sulla posta elettronica". A large blue button at the bottom contains the text "Recupero".

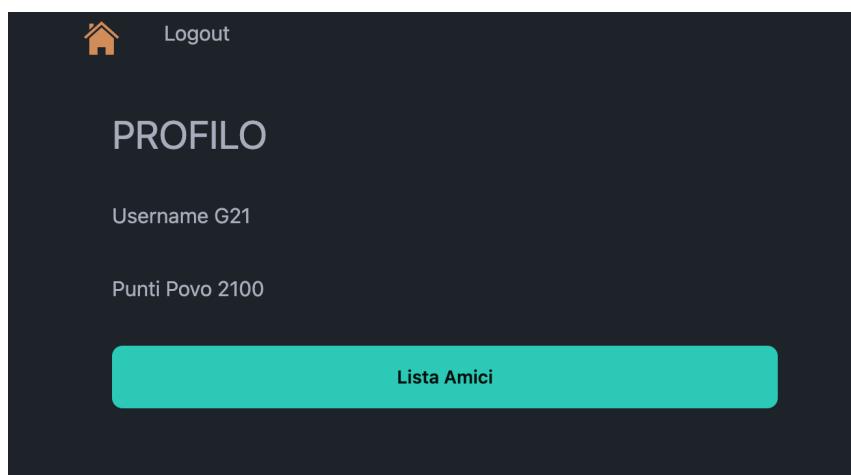
### 3.0.5 Logout

Schermata per il logout.



### 3.1 Profilo utente

Schermata di visualizzazione del profilo utente.



### 3.2 Lista amici

Schermata di visualizzazione della propria lista amici.

Nickname	Punti	Numero
<u>marcostrada</u>	827	1234567890
<u>matejdelcoco</u>	1366	0987654321
<u>user6</u>	1500	1234567890

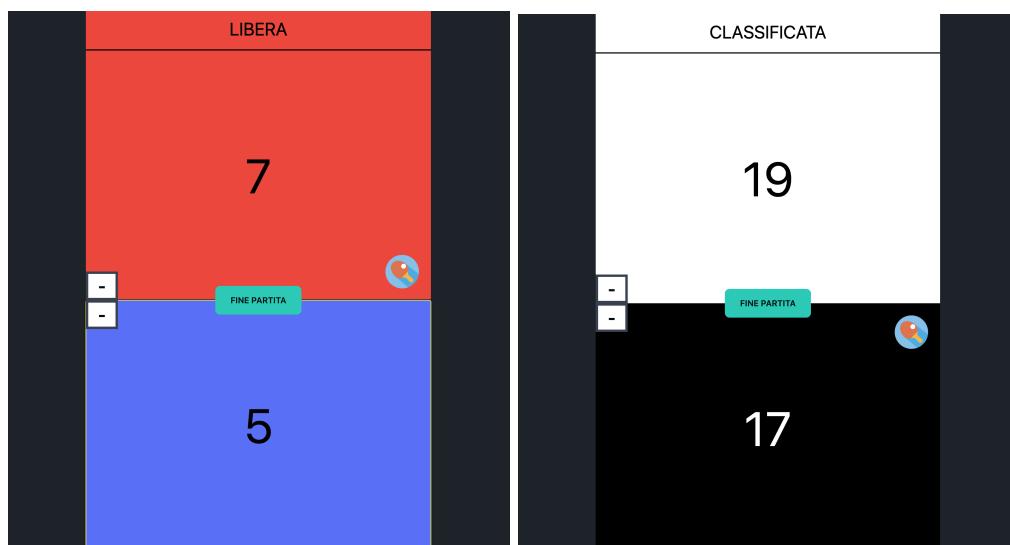
### 3.2.1 Scegli modalità

Schermata per scegliere la tipologia e modalità di partita all'interno della funzione conta-punti.



### 3.2.2 Conta punti

Schermate di conta-punti, rispettivamente per partite libere e classificate.



### 3.2.3 Aggiungi partita

Schermata per aggiungere una partita al sistema e nelle classifiche.

AGGIUNGI PARTITA

G21

21

Utente sconfitto :(

7

Salva

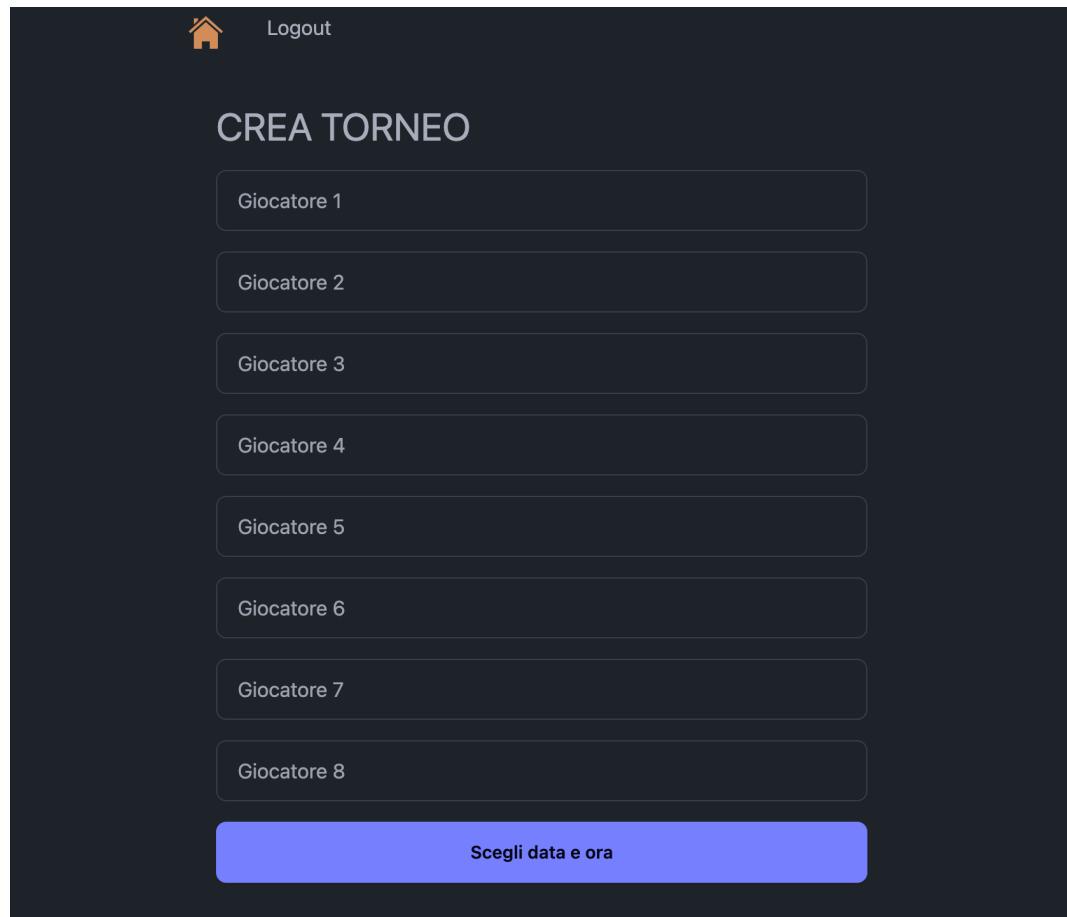
### 3.3 Classifica

Schermata per visualizzare la classifica della stagione attuale.

Rank	Username	Punti
1	<u>G21</u>	2100
2	<u>user6</u>	1500
3	<u>matejdelcoco</u>	1366
4	<u>user2</u>	1300
5	<u>user4</u>	1000
6	<u>user5</u>	1000
7	<u>user1</u>	1000
8	<u>user3</u>	932
9	<u>marcostrada</u>	827

### 3.3.1 Crea torneo

Schermata per la creazione di un nuovo torneo.



## 4 Esecuzione del codice

### 4.1 Esecuzione in locale

Istruzioni per eseguire PingPovo in **localhost**.

Hostare il **front-end**:

- clonare la repo da GitHub
- aprire la cartella di root da terminale ed eseguire `$npm install, $npm run dev` .

Hostare il **back-end**:

- clonare la repo da GitHub
- aprire la cartella di root da terminale ed eseguire `$npm install, $npm run autodev` .

Saranno visibili sulle porte:

- `http://localhost:5173`
- `http://localhost:9999`

Per provare le funzionalità dell'utente autenticato, creare un nuovo account o utilizzare le seguenti credenziali:

username: G21

password: g21

## 5 Testing

Per la fase di testing abbiamo deciso di utilizzare Vitest per evitare i alcuni problemi di compatibilità del nostro progetto con Jest.

```
const base_url = "http://localhost:9999/api";
💡
test('Login corretto', async () => {
    const username = "matejdelcoco";
    const password = "matej10";

    const res = await fetch(base_url + "/login", {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify({ username, password })
    });
    expect(res.status).toBe(200);
});

test('Login non corretto, deve fallire e ritornare "unauthorized"', async () => {
    const username = "matejdelcoco";
    const password = "matej11";

    const res = await fetch(base_url + "/login", {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify({ username, password })
    });
    expect(res.status).toBe(403);
});
```

### 5.1 Risultato test login

```
> jest
PASS  source/test.ts
  ✓ Login corretto (337 ms)
  ✓ Login non corretto, deve fallire e ritornare "unauthorized" (92 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:  0 total
Time:        0.6 s, estimated 1 s
Ran all test suites.
```

Come si vede dall'immagine i 2 test del login sono stati passati con successo.

DELIVERABLE 4, GRUPPO G21: Marco Strada e Matej Del Coco