📌Riepilogo Autenticazione con NextAuth.js + Prisma

Questo file descrive la struttura del database e il flusso di autenticazione quando si usa **Next.js + NextAuth.js + Prisma**.

🕰️Struttura del Database (schema.prisma)

```
model User {
  id            String   @id @default(cuid())
  name          String?
  email         String?  @unique
  emailVerified DateTime?
  image         String?
  password      String?   // hash della password se usi email/password
  accounts      Account[]
  sessions      Session[]
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
}

model Account {
  id                String  @id @default(cuid())
  userId            String
  type              String
  provider          String
  providerAccountId String
  refresh_token     String?
  access_token      String?
  expires_at        Int?
  token_type        String?
  scope             String?
  id_token          String?
  session_state     String?

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@unique([provider, providerAccountId])
}

model Session {
  id           String   @id @default(cuid())
  sessionToken String   @unique
  userId       String
  expires      DateTime
```

```
  user User @relation(fields: [userId], references: [id], onDelete: Cascade)
}

model VerificationToken {
  identifier String
  token      String   @unique
  expires    DateTime

  @@unique([identifier, token])
}
```

🕰️Funzione di ogni tabella

`User`

- Contiene i dati base dell'utente.
- Campi principali: `email`, `password` (se usi login tradizionale), `name`, `image`.
- Relazioni: più `accounts` (OAuth) e più `sessions`.

`Account`

- Contiene i provider esterni collegati (OAuth).
- Campi principali: `provider` (es. google, github), `providerAccountId` (ID del provider), token di accesso.
- Permette login multipli sullo stesso `User`.

`Session`

- Contiene le sessioni utente attive.
- Campi principali: `sessionToken`, `expires`.
- Usata se la session strategy è `database`.

`VerificationToken`

- Serve per login via **magic link** o reset password.
- Campi principali: `identifier` (email), `token` (univoco), `expires`.

🕰️Providers tipici

Nel file `pages/api/auth/[...nextauth].ts` puoi configurare: - **OAuth** → GitHub, Google, ecc. - **Credentials** → email/password gestite manualmente. - **Email provider** → magic link con `VerificationToken`.

Esempio:

```
import NextAuth from "next-auth";
import GitHubProvider from "next-auth/providers/github";
```

```
import GoogleProvider from "next-auth/providers/google";
import CredentialsProvider from "next-auth/providers/credentials";

export default NextAuth({
  providers: [
    GitHubProvider({ clientId: process.env.GITHUB_ID!, clientSecret:
process.env.GITHUB_SECRET! }),
    GoogleProvider({ clientId: process.env.GOOGLE_ID!, clientSecret:
process.env.GOOGLE_SECRET! }),
    CredentialsProvider({
      name: "Credentials",
      credentials: { email: { type: "email" }, password: { type:
"password" } },
      async authorize(credentials, req) {
        // Trova utente nel DB e verifica la password hashata
      }
    })
  ]
});
```

🕰️ Flusso di Autenticazione

1. **Registrazione con email/password** → Crei utente in `User` con `password` hashata.
2. **Login con email/password** → Controlli se l'utente esiste e verifichi hash.
3. **Login con OAuth (Google/GitHub)** → Viene creata una entry in `Account` collegata al `User`.
4. **Sessione** → A seconda della config, può essere salvata in `Session` o gestita con JWT.
5. **Magic link o reset password** → Usa tabella `VerificationToken`.

🕰️ In sintesi

- `User` = info utente
- `Account` = login esterni (OAuth)
- `Session` = sessioni persistenti (se non JWT)
- `VerificationToken` = magic link / reset

Questo schema è flessibile e ti permette di supportare **sia credenziali tradizionali, sia OAuth, sia magic link**.