Die Quiz-App ist in drei wesentliche Teile aufgeteilt:

1. Server

- Startpunkt: `src/server.js`
- Aufgaben:
  - Bereitstellung von HTML/JS-Dateien (statische Inhalte)
  - REST-API-Endpunkte für z. B. Fragen, Kategorien, neue Sessions
  - Verwaltung der Sessions (Code, Fragen, Antworten)
  - WebSocket-Schnittstelle für Live-Kommunikation

◇ 2. Client (Teilnehmer)

- Datei: `client.html` + `client.js`
- Ablauf:
  - Nutzer gibt Quiz-Code ein → Verbindung zum Server via WebSocket
  - Fragen werden vom Server gesendet (`show-question`)
  - Antwort wird gesendet (`submit-answer`)
  - Anzeige von „Quiz beendet“ oder korrekter Antwort

◇ 3. Host/Admin

- Datei: `host.html` + `host.js`
- Ablauf:
  - Startet ein neues Quiz über Kategorie-Auswahl
  - Server generiert Quiz-Code + Fragen
  - Kann jede Frage einzeln „freigeben“ (→ `next-question`)
  - Sieht Live-Antworten + zeigt später das Ergebnis an (`show-results`)

1. Server (Node.js mit Express & WebSockets)

Verwendete Technologien

- **Node.js**: Serverseitige JavaScript-Plattform
- **Express.js**: Webframework für REST-APIs
- **Socket.IO**: Echtzeitkommunikation via WebSockets
- **Prisma**: Objekt-Relationales Mapping (ORM) für den Datenbankzugriff
- **HTML/CSS/JS**: Für die Client-Oberflächen (Admin & Teilnehmer)

Statische Dateien

- Der Server stellt alles im Ordner `src/public/` zur Verfügung:

```
app.use(express.static('src/public'));
```

WebSocket Server

- Socket.IO wird über den HTTP-Server verbunden:

```
const io = new Server(server);
```

Starten des Servers

```
node src/server.js
```

2. REST-API-Endpunkte

Methode	Pfad	Beschreibung
GET	/api/categories	Liefert alle verfügbaren Kategorien
GET	/api/local-ip	Gibt die lokale IP-Adresse zurück
GET	/api/client-count/:code	Anzahl verbundener Clients für ein Quiz
POST	/api/generate-code	Erstellt neue Quiz-Session mit zufälligem 6-stelligem Code
GET	/api/questions	Holt alle Fragen (für Adminbereich)
POST	/api/questions	Neue Frage speichern
PUT	/api/questions/:id	Frage aktualisieren

3.WebSocket-Kommunikation (mit Socket.IO)

3.1 Client-Events (`src/public/src/client.js`)

Art	Eventname	Beschreibung
emit	join-quiz	Schickt den 6-stelligen Quizcode an den Server, um dem Spiel beizutreten.
on	show-question	Der Server sendet eine neue Frage an alle verbundenen Clients.
emit	submit-answer	Der Client sendet die Antwort (z. B. "A", "B", ...) an den Server.

3.2 Emit-Host-Events (`src/public/src/host.js`)

Art	Eventname	Beschreibung
`emit`	`join-quiz`	Host tritt ebenfalls einem Raum bei (zur Visualisierung).
`emit`	`show-question`	Startet die nächste Frage und sendet sie an alle Clients.
`emit`	`Next-question`	Startet die nächste Frage im Quiz.
`emit`	`Show-results`	Signalisiert dem Server, die richtige Antwort zu zeigen.

3.3 On-Host-Events (`src/public/src/host.js`)

Eventname	Nutzdaten	Zweck
`joined-count`	`count`	Zeigt aktuelle Anzahl der Teilnehmer
`show-question`	`text`, `optionA–optionD`	Frage anzeigen + Antwortmöglichkeiten setzen
`answer-stat`	`questionId`, `option`	Erhöht Zähler für gegebene Antworten
`show-results`	`correct`	Hebt die richtige Antwort visuell hervor

4. Datenmodell (Datenbank über Prisma)

Tabelle: question

Feld	Typ	Bedeutung
`id`	Integer	Eindeutige ID
`category`	String	Kategorie wie „Allgemeinwissen“
`question`	String	Frage-Text
`correctAnswer`	String	Die jeweils richtige Antwort z. B. „A“
`answers`	String[]	Antwortmöglichkeiten (A–D)