Meilenstein 2 WebE – Dennis Appel & Luca Malisan

Netzwerkprotokoll

Netzwerkprotokoll-Dokumentation

Dieses Protokoll definiert die Kommunikation zwischen dem Spielclient und dem Server für das Multiplayer-Cookie-Clicker-Virus-Spiel. Es verwendet JSON als Nachrichtenformat.

Nachrichtentypen Game Session

1. ‘create-session’
   - Zweck: Wird verwendet, um eine Spiel-Session zu erstellen.
   - Richtung: Client -> Server

   Payload:
   ```
   {
     ‘event’: ‘session-creation-successful’,
     ‘data’: ‘45’
   }
   ```
2. ‘session-creation-successful’
   - Zweck: Wird verwendet, um Client zu informieren, dass Session erstellt wurde.
   - Richtung: Server -> Client

   Payload:
   ```
   {
     ‘event’: ‘session-creation-successful’,
     ‘data’: ‘#a3f1e9’ // Session Code
   }
   ```

3. ‘ready-for-game-start’
   - Zweck: Wird verwendet, um zu bestätigen, dass Client bereit ist.
   - Richtung: Client -> Server

   Payload:
   ```
   {
     ‘event’: ‘session-creation-successful’,
     ‘data’: ‘’
   }
   ```

4. ‘start-timer’
   - Zweck: Server bestätigt, dass er von allen Clients ein ‘ready-for-game-start’ Message erhalten hat und der Timer client-seitig gestartet werden kann.
   - Richtung: Server -> Client

   Payload:
   ```
   {
   ```

‘event’: ‘start-timer’,
’data’: ‘45’ // Duration
}

5. ‘stop-session’
   - Zweck: Wird verwendet, um die Session zu beenden, wenn der server-seitige Timer abgelaufen ist
   - Richtung: Server -> Client

   Antwortbeispiel:
   {
   ‘event’: ’stop-session’,
   ‘data’: ‘’
   }

6. ‘get-session-info’
   - Zweck: Wird verwendet, um Informationen zur aktuellen Session zu erhalten
   - Richtung: Client -> Server

   Payload
   {
   ‘event’: ‘get-session-info’
   ‘data’: ‘’
   }

   Antwortbeispiel:
   {
   ‘sessionKey’: ‘#a3f1e9’,
   ‘joinedPlayers’: [‘H4ckerman’, ‘H4ckerman2’],
   ‘admin’: true
   }

7. ‘join-session’
   - Zweck: Wird verwendet, um einer Session beizutreten
   - Richtung: Client -> Server

   Payload:
   {
   ‘event’: ‘join-session’
   ‘data’: ‘#a3f1e9’ //session key
   }

8. ‘join-successful’
   - Zweck: Wird verwendet, um den Client zu informieren, dass er erfolgreich gejoint ist
   - Richtung: Server -> Client

   Payload:
   {

‘event’: ‘join-successful’,
‘data’: ‘’
}

Nachrichtentypen Gamepoints

1. ‘handle-button-clicks’
   - Zweck: Dient dazu, den Users bei Clicks auf den Button Punkte (Viren) zu geben
   - Richtung: Client -> Server

   Payload:
   {
   ‘event’: ‘handle-button-clicks’
   ‘data’: ‘’
   }

   Antwortbeispiel:
   {
   ‘data’: ‘555’ //current virus amount of user
   }

2. ‘leaderboard’
   - Zweck: Dient dazu, die Clients über Änderungen am Leaderboard zu informieren
   - Richtung: Server -> Client

   Payload:
   {
   ‘event’: ‘leaderboard’,
   ‘data’: [
   {
   ‘userName’: ‘H4ckerman’,
   ‘points’: 1337
   },
   {
   ‘userName’: ‘H4ckerman2’,
   ‘points’: 666
   }
   ]
   }

Nachrichtentypen Chat

1. ‘chat-message’ (Client -> Server)
   - Zweck: Dient dem Versenden von Chatnachrichten und persistiert die Nachrichten in der DB.
   - Richtung: Client -> Server

   Payload:
   {

‘event’: ‘chat-message,
‘data’: {
‘message: ‘I’ve hacked the system!’
}
}

2. ‘chat-message’ (Server -> Client)

- Zweck: Server schickt erhaltene Chat-Nachricht an alle anderen Clients weiter

Payload:
{
‘event’: ‘chat-message,
‘data’: {
‘message’: ‘I’ve hacked the system!’
‘username’: ‘H4ckerman’
}
}

Nachrichtentypen Authentication

1. ‘register-user’
- Zweck: Dient dazu einen Nutzer zu registrieren, damit dieser sich einloggen kann
- Richtung: Client -> Server

Payload:
{
‘event’: ‘register-user’,
‘data’: {
‘username’: ‘H4ckerman’
‘password’: ‘mypass’
}
}

2. ‘registration-successful’
- Zweck: Informiert den Client, dass Registrierung erfolgreich verlief
- Richtung: Server -> Client

Payload:
{
‘event’: ‘registration-successful’,
‘data’: ‘’
}

3. ‘login-user’
- Zweck: Dient dazu einen User mit Username und Passwort einzuloggen
- Richtung: Client -> Server

Payload:
{

‘event’: ‘login-user’,
‘data’: {
‘username’: ‘H4ckerman’
‘password’: ‘mypass’
}
}

4. ‘login-successful’
   - Zweck: Dient dazu den Client zu informieren, dass der User authentifiziert ist
   - Richtung: Server -> Client

   Payload:
   {
   ‘event’: ‘login-successful’,
   ‘data’: {
   ‘jwt’: ‘abc…’,
   ‘refresh-token’: ‘def…’
   }

5. ‘register’

   - Zweck: Dient dazu die Client-Applikation beim Server zu registrieren
   - Richtung: Client -> Server

   Payload:
   {
   ‘event’: ‘register’,
   ‘data’: ‘’
   }

   Antwortbeispiel:
   {
   ‘jwt’: ‘abc…’,
   ‘refresh-token’: ‘def…’
   }

Aufbau und Funktion Server

Der Server besteht aus dem NestJS-Backend (Version 11) und einer Postgres-Datenbank. NestJS verwendet einen modularen Aufbau, wobei jedes Package ähnlich strukturiert ist:

- ***.gateway.ts***: enthält die Methoden der Websocket-Schnittstellen
- ***.service.ts**:* enthält Funktionen für Datenbankabfragen
- ***.module.ts*****:** definiert die externen Imports und macht die Modulklassen für die anderen Packages verfügbar

Der Server besteht aus folgenden Modulen:

- ***config***: enthält die Konfiguration für die Datenbankanbindung
- ***auth***: zuständig für das Login und die Registrierung von Usern
- ***chat*****:** zuständig für die Chat-Funktion
- ***game-session:*** zuständig für die Erstellung von Game-Sessions, den Beitritt anderer Spieler und den Spielstart.
- ***Model:*** bildet die Datenbanktabellen mittels TypeORM ab
    - ***chatMessage***: Protokollierung von Chatnachrichten mit Referenz auf den User (Autor) und die GameSession
    - ***effect***: Effekte, die erworben werden können
    - ***gameSession***: Erstellte GameSessions mit Referenz auf den User (Ersteller)
    - ***user***: Registrierte Benutzer
    - ***userActionLog***: Protokoll aller Spieleraktionen mit Referenz auf UserGameSession (Spieler und GameSession)
    - ***userEffect***: Effekte, von denen ein User in der aktuellen GameSession betroffen ist, Referenz auf User und Effekt
    - ***userGameSession***: Protokoll, welcher User an welcher GameSession teilnimmt oder teilgenommen hat, Referenz auf User und GameSession
- ***static***: globale Variablen, z.B. Liste aller registrierten Sockets
- ***users***: Datenbankfunktionen für User-Entity
- ***app.gateway.ts***: Websocket-Schnittstelle für die Registrierung des Clients beim Server
- ***app.module.ts***: Root-Modul, das alle anderen Module importiert
- ***main.ts***: Start-Klasse

Aufbau und Funktion Client

Der Client ist in Angular 19.2.0 geschrieben. Angular verwendet eine modulare Struktur, wobei jedes Package ähnlich aufgebaut ist:

- ***.component.html***: HTML Template des Moduls
- ***.component.css:*** Stylesheet für das Modul
- ***.component.ts***: Zuständig für die Dynamik des Frontends, z.B. Template Variablen, Form Submissions, Event Listeners etc.

Der Client besteht aus folgenden Modulen:

- ***chat:*** Chat-Funktion
- ***game:*** Orchestierung der Komponenten ***Chat***, ***Leaderboard***, ***ShopPreview*** und ***Timer***
- ***gameLoading:*** Ladescreen zwischen Session Joining und Game-Start
- ***leaderboard:*** Game-Komponente, die aktuelle Rangliste der Spieler in der Session anzeigt
- ***login:*** Login-Screen, falls kein gültiges JWT vorhanden
- ***register:*** Registrierungsformular für neue User
- ***sessionCreation:*** Formular, um neue Sessions zu erstellen
- ***sessionJoining:*** Formular, um Sessions beizutreten
- ***shopPreview:*** Game-Komponente für PowerUp-Shop
- ***timer:*** Game-Komponente, die die verbleibende Spielzeit anzeigt
- ***app.component.****: Root-Komponente
- ***app-module.ts***: Root-Modul
- ***app-routing.module.ts:*** Zuweisung von Angular-Komponenten und eindeutigen URLs
- ***core.service.ts***: enthält globale Logik, z.B. Registierung des Clients beim Server

Kommunikation Client - Server

Anmeldung Client beim Server

Client		***Server***
****.component.ts***	***core.service.ts***	***app.gateway.ts***
	Initialisierung des Observables «initialized»	
Subscription auf «initialized»		
	Initialisierung socket.io und Listener auf connect-Event	
		Connection akzeptiert
	• Reaktion auf connect-event • Aufrufen von Websocket-Route «register» und Senden des jwt-Tokens	
		• Validierung des JWT-Tokens • Ggf. Reaktivierung per Refresh-Token • bei Erfolg Speicherung des Sockets als authentifiziert • Senden der Response
	Überprüfung der Response • Success = true: «initialized» wird auf true gesetzt • Success = false: Weiterleitung auf login-page	
• Reaktion auf Value Change von «initialized» • Ausführung eigener Logik		

Chat-Funktion

Client	***Server***
chat.component.ts	***chat.gateway.ts***
Event Listener auf «Senden-Button» hinzufügen	
Listener auf WebSocket-Event «chat-message» hinzufügen	
Reaktion auf Event Listener: • Geschriebene Chat-Nachricht auslesen • Aufrufen von WebSocket-Route «chat-message»	
	• GameSession anhand Client Socket auslesen • Speicherung der Chat-Nachricht in der DB, gemappt auf User und GameSession • User anhand Client Socket auslesen • Chat-Nachricht und Username des Verassers an alle registrierten Sockets mit Betreff «chat-message» schicken
Reaktion auf Listener auf Route «chat-message»: • Hinzufügen der Nachricht in lokale Array aller Chat-Nachrichten. • Abbildung des Arrays der Chat-Nachrichten im Template	