Zahlen Ratespiel 2

Einführung & Ziele

Das erste Beispiel war eine komplette Client-Seitige Implementierung mit einigen Nachteilen: Findige Endanwender können über die Entwicklertools die Zahl, die zu erraten ist, entdecken. Die einzige wirkliche sichere Variante ist es, diese geheime Zahl nicht am Client zu speichern. Anstatt eine Zufallszahl am Client zu generieren, rufen wir eine Methode am Server auf um ein neues Spiel anzulegen. Auch die Logik zum Vergleich des Rateversuches mit der geheimen Zahl verschieben wir in einen Aufruf am Server und geben nur das Ergebnis für die Anzeige zurück.

1. Erweitern der Server Logik

Wir erweitern server.js um 2 Methoden, eine als POST Aktion '/newgame' welche ein neues Spiel anlegt und eine als GET Aktion '/try' die zwei Parameter nimmt. '/newgame' legt ein neues Spiel Objekt an, fügt es in das globale 'game' Array ein und gibt die id (= Index des Spiels) des Spieles zurück:

server.js angularjs\assignments\numbergame2

```
1  var express = require('express');
2  var app = express();
3  app.use(express.static('./'));
4
5  var games = [];
6  app.post('/newgame', function(req, res) {
7    var newGame = {
8      id: games.length,
9      theNumber: Math.floor(Math.random() * 100),
10     guessCount: 0
11   };
12
13   games.push(newGame);
14   res.send(JSON.stringify({ id: newGame.id, guessCount: newGame.guessCount }));
15 });
```

Die 'try' Methode erwartet zwei URI Parameter die in der URL durch ':id' und ':guess' angegeben sind und über req.params.id und req.params.guess zugegriffen werden können. Die Methode greift auf das Spiel Objekt im globalen 'games' array zu, überprüft die guess Variable gegen die im 'game' Objekt gespeicherte Zahl (theNumber):

server.js angularjs\assignments\numbergame2

```
16
17 app.get('/try/:id/:guess', function(req, res) {
18   var gameId = req.params.id;
19   var guess = req.params.guess;
20   var game = games[gameId];
21   game.guessCount++;
22
23   if (game.theNumber == guess) {
24     console.log('game ' + game.id + ': finished!');
25     game.finished = true;
26     res.send(JSON.stringify({ result: 'correct', guessCount: game.guessCount, theNumber: game.theNumber }));
27   } else if (guess > game.theNumber) {
28     console.log('game ' + game.id + ': ' + guess + ' is too high!');
29     res.send(JSON.stringify({ result: 'toohigh', guessCount: game.guessCount }));
30   } else if (guess < game.theNumber) {
31     console.log('game ' + game.id + ': ' + guess + ' is too high!');
32     res.send(JSON.stringify({ result: 'toolow', guessCount: game.guessCount }));
33   }
34 });
```

2. Abändern des Controllers

Damit die Methoden am Server aufgerufen werden, müssen wir den Controller so umbauen, dass bei der Initialisierung ein Aufruf zur ‚/newgame' Methode ausgeführt wird. Dafür verwenden wir das $http service, das wir über Dependency Injection im Controller erhalten.

Um später leichter ein neues Spiel starten zu können ohne die Seite neu laden zu müssen kapseln wir die Initialisierung in eine eigene methode 'init'. Diese setzt die initialen Variablen, führt schließlich den POST Aufruf an '/newgame' auf und registriert einen 'success' (der erste Parameter für then) callback der aus dem response.data objekt die variablen für das Spiel ausliest:

numbergame.js angularjs\assignments\numbergame2

```
1  var app = angular.module('numbergameApp', []);
2  app.controller('numbergameController', ['$scope', '$http', function($scope, $http) {
3      $scope.init = function() {
4          $scope.currentGuess = null;
5          $scope.result = null;
6          $scope.guessCount = 0;
7          $scope.gameId = -1;
8
9          $http.post('/newgame')
10         .then(function(response) {
11             $scope.gameId = response.data.id;
12             $scope.guessCount = response.data.guessCount;
13         },
14         function(response) {
15             /* error */
16         });
17     };
18     $scope.init();
```

Weiters modifizieren wir die check Methode, die anstatt das Ergebnis zu vergleichen, den Aufruf über einen $http.get Aufruf weiterleitet:

numbergame.js angularjs\assignments\numbergame2

```
20      $scope.isChecking = false;
21      $scope.check = function() {
22          $scope.isChecking = true;
23          $http.get('/try/' + $scope.gameId + '/' + $scope.currentGuess)
24          .then(function(response) {
25              $scope.isChecking = false;
26              $scope.currentGuess = null;
27              $scope.guessCount = response.data.guessCount;
28              $scope.result = response.data.result;
29              if ($scope.result == 'correct') {
30                  $scope.theNumber = response.data.theNumber;
31              }
32          },
33          function() {
34              /* error */
35              $scope.isChecking = false;
36          });
37      }
38  }]);
```

3. UI Verbesserungen

Da wir nun asynchrone $http Operationen verwenden, sollte man den Benutzer darauf hinweisen, wenn eine Operation noch im Gange ist. Dies kann entweder über einen eigenen Indikator ('BusyIndicator') oder über schlichtes Deaktivieren von Controls geschehen. Wir verwenden dafür ng-disabled in der UI:

numbergame.html angularjs\assignments\numbergame2

```
24          <form class="form-inline">
25              <div class="form-group">
26                  <input class="form-control" ng-disabled="gameId == -1 || isChecking" type="number" ng-model="currentGuess">
27              </div>
28              <button class="btn btn-primary" ng-disabled="gameId == -1 || isChecking" ng-click="check()">Check</button>
29          </form>
```

4. Neues Spiel Starten

Eine weitere Verbesserung ist es ein neues Spiel starten zu können. Dies ist durch Hinzufügen eines neuen <button> Elementes leicht möglich. Auch wollen wir verhindern, dass der Spieler, weitere Versuche durchführen kann, wenn er ein Spiel erfolgreich beendet hat:

numbergame.html angularjs\assignments\numbergame2

```
24          <form class="form-inline">
25              <div class="form-group">
26                  <input class="form-control"
27                          ng-disabled="gameId == -1 || isChecking"
28                          type="number" ng-model="currentGuess">
29              </div>
30              <button class="btn btn-primary"
31                      ng-disabled="gameId == -1 || isChecking || result == 'correct'"
32                      ng-click="check()">Check</button>
33
34              <button class="btn btn-default"
35                      ng-disabled="gameId == -1 || isChecking"
36                      ng-click="init()">New Game!</button>
37          </form>
```

5. Zeitnehmung

Um das Spiel etwas motivierender zu machen, wäre es spannend nicht nur die Anzahl der Versuche sondern auch eine Zeitmessung einzuführen. Dafür können wir das $interval Service von AngularJS verwenden. Mit $interval kann man Funktionen registrieren die in regelmäßigen Abständen aufgerufen werden. Beim Starten eines neuen Spieles, merken wir uns also die aktuelle Uhrzeit und registrieren einen Callback der alle 30 Millisekunden die vergangene Zeit aktualisiert. Sobald das Spiel beendet wurde (oder ein neues gestartet wurde), rufen wir die $interval.cancel Methode auf um die Intervalaufrufe zu beenden. Für die bessere Struktur des Codes, kapseln wir das starten und stoppen der Zeitnehmung in zwei Methoden.

numbergame.js angularjs\assignments\numbergame2

```
1  var app = angular.module('numbergameApp', []);
2  app.controller('numbergameController',
3  ['$scope', '$http', '$interval',
4  function($scope, $http, $interval) {
5      $scope.stopTimer = function() {
6          if ($scope.gameTimer) {
7              $interval.cancel($scope.gameTimer);
8              $scope.gameTimer = null;
9          }
10     };
11
12     $scope.startTimer = function() {
13         $scope.start = new Date().getTime();
14         $scope.elapsed = 0;
15         $scope.gameTimer = $interval(function() {
16             $scope.elapsed = (new Date().getTime()) - $scope.start;
17         }, 30);
18     };
```

numbergame.js angularjs\assignments\numbergame2

```
19
20      $scope.init = function() {
21          $scope.stopTimer();
22          $scope.currentGuess = null;
23          $scope.result = null;
24          $scope.guessCount = 0;
25          $scope.gameId = -1;
26          $scope.elapsed = 0;
27
28          $http.post('/newgame')
29          .then(function(response) {
30              $scope.gameId = response.data.id;
31              $scope.guessCount = response.data.guessCount;
32              $scope.startTimer();
33          },
34          function(response) {
35              /* error */
36          });
37      };
38      $scope.init();
39
40      $scope.isChecking = false;
41      $scope.check = function() {
42          $scope.isChecking = true;
43          $http.get('/try/' + $scope.gameId + '/' + $scope.currentGuess)
44          .then(function(response) {
45              $scope.isChecking = false;
46              $scope.currentGuess = null;
47              $scope.guessCount = response.data.guessCount;
48              $scope.result = response.data.result;
49              if ($scope.result == 'correct') {
50                  $scope.theNumber = response.data.theNumber;
51                  $scope.stopTimer();
52              }
53          },
54          function() {
55              /* error */
56              $scope.isChecking = false;
57          });
58      }
59  }]);
```

Schließlich benötigen wir im HTML noch ein Element zur Darstellung der Zeit:

numbergame.html angularjs\assignments\numbergame2

```
15          <div>
16              Guess a number between 0 and 100
17          </div>
18
19          <div>
20              Number of tries: {{guessCount}}
21          </div>
22          <div>
23              Elapsed Time: {{elapsed}} ms
24          </div>
```