

Loopback chat service demo

Nulladik lépés

1. Mongo DB telepítése
 - a. Indítás: `mongod --dbpath c:\Tools\MongoDB\Data`
2. Webstorm indítás
3. CMD indítás

Projekt inicializálás és modellek definiálása

1. Új projekt létrehozása Webstormban
2. Projekt inicializálása:
 - a. `slc loopback`
 - b. Kiírt parancsok követése
 - c. Projekt mappa megnyitása
 - d. `npm install` futtatása
 - e. Project Settingsben node_modules exclude
3. Nézzük át a generált dolgokat
4. `slc arc`
 - a. Indulás után a kapott url-t bemásolni a böngészőbe (ez megnyílik magától is, ha újra kell indítani az arc-ot akkor megváltozhat a port!)
 - b. Bejelentkezés
5. Adatforrás konfiguráció (composer)
 - a. `npm install --save loopback-connector-mongodb`
 - b. Utána composer / DataSources
 - i. Név: mongoDev
 - ii. Url: `mongodb://localhost/`
 - iii. Database: chatdemo
 - iv. Connector MongoDB
6. Modellek definiálása -> Composer
 - a. Room
 - i. Name required
 - b. Message
 - i. Text required
7. Nézzük meg mi generálódott
 - a. **idInjection true**, beállítása
8. Nézzük meg a model-config.json-t
 - a. Adatforrásokat állítsuk be!
9. Indítsuk el
 - a. `slc run`
 - b. Explorer url megnyitása
 - c. User modellt nézzük meg!

Modell kapcsolatok létrehozása

1. `slc loopback:model` megmutatása
 - a. Ezt a modellt töröljük!
2. Room és Message közötti kapcsolat megadása

```

Adam@SZADAM-WORK C:\Autsoft\Projects\LoopbackChatServerDemo
> slc loopback:relation
? Select the model to create the relationship from: Room
? Relation type: has many
? Choose a model to create a relationship with: Message
? Enter the property name for the relation: messages
? Optionally enter a custom foreign key: roomId
? Require a through model? No

Adam@SZADAM-WORK C:\Autsoft\Projects\LoopbackChatServerDemo
> slc loopback:relation
? Select the model to create the relationship from: Message
? Relation type: belongs to
? Choose a model to create a relationship with: Room
? Enter the property name for the relation: room
? Optionally enter a custom foreign key: roomId

```

3. Nézzük meg mi történt
4. Próba: *slc run*
 - a. Hozzunk létre Roomot
 - b. Hozzunk létre 2 Message-et roomhoz
 - c. Próbáljuk lekérni a Room message-eit
 - d. Kérjük le a Message-eket room függetlenül
 - e. Kérjük le a Message-eket room függetlenül, betöltve a room adatokat:
Filter = {"include": "room"}
5. Message és User kapcsolat

```

Adam@SZADAM-WORK C:\Autsoft\Projects\LoopbackChatServerDemo
> slc loopback:relation
? Select the model to create the relationship from: Message
? Relation type: belongs to
? Choose a model to create a relationship with: User
? Enter the property name for the relation: author
? Optionally enter a custom foreign key: authorId

Adam@SZADAM-WORK C:\Autsoft\Projects\LoopbackChatServerDemo
> slc loopback:relation
? Select the model to create the relationship from: User
? Relation type: has many
? Choose a model to create a relationship with: Message
? Enter the property name for the relation: messages
? Optionally enter a custom foreign key: authorId
? Require a through model? No

```

6. Nézzük meg mi történt
7. Próba: *slc run*
 - f. Regisztráljunk egy User-t:

```

{
  "email": "admin@test.com",
  "password": "123456"
}

```
8. Lépjünk be
9. Hozzunk létre üzenetet
 - a. Nem kerül be a bejelentkezett user id-je!
 - b. Adjuk hozzá kézzel

10. Legyen ez automatikus

- a. Definiáljunk egy *before save operation hook*-ot
- b. Step 1: message.js-be:

```
Message.observe('before save', function(ctx, next) {  
  console.log("Message before save called");  
});
```

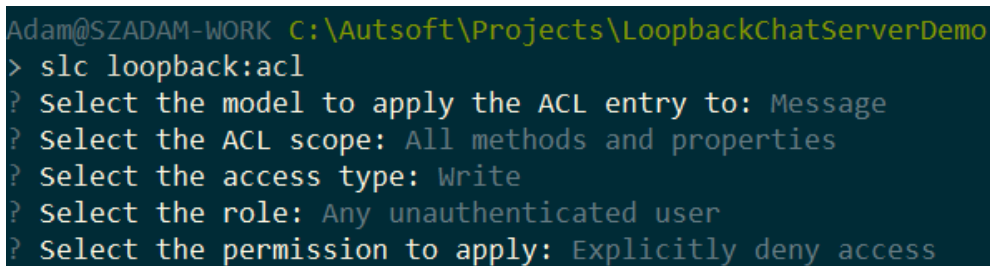
- c. Step 2: Próba, nem fog semmi történni, csak a logban látjuk ezt az üzenetet, azért mert a next nincs meghívva
- d. Step 3: Implementálni

```
var message = ctx.instance;  
var accessToken = loopback.getCurrentContext().get("accessToken");  
if (!message.authorId && accessToken) {  
  message.authorId = accessToken.userId;  
}  
next(null, message);
```

- e. Próba, bejelentkezett felhasználóval!

Jogosultság beállítások

1. Csak bejelentkezett felhasználó írhat kommentet:
 - a. Parancs: *slc loopback:acl*



```
Adam@SZADAM-WORK C:\Autsoft\Projects\LoopbackChatServerDemo  
> slc loopback:acl  
? Select the model to apply the ACL entry to: Message  
? Select the ACL scope: All methods and properties  
? Select the access type: Write  
? Select the role: Any unauthenticated user  
? Select the permission to apply: Explicitly deny access
```

- b. Nézzük meg mi történt a *message.json*-ban
- c. Próba bejelentkezett felhasználó nélkül
 - i. Itt a room/message menni fog, azért mert erre külön a Roomban is be kell állítani!
 - ii. Próbáljuk ki a POST Message kérést!
 - iii. Bejelentkezve próbáljuk ezt ki újra
- d. Állítsunk be ACL-t a *room.json* ban is

```
{  
  "accessType": "WRITE",  
  "principalType": "ROLE",  
  "principalId": "$unauthenticated",  
  "permission": "DENY",  
  "property": [  
    "__create_messages",  
    "__delete_messages",  
    "__destroyById_messages",  
    "__updateById_messages"  
  ]  
}
```

- e. Próbáljuk ki újra

2. Használjunk szerepköröket!

- a. Adatbázis seed írás, ahol létrehozuk a szerepkört és egy admin felhasználót, ezeket mappel-jük is össze
- b. Hozzuk létre a *seed.js*-t a boot mappába
- c. Töltsük le a fájl vázát innen: TODO
- d. Telepítsük fel az async libraryt: *npm install async --save*
- e. Szerezzünk refenciát az async libre

f. Írjuk meg a kéréseket:

```
async.parallel({
  adminUser: function(callback) {
    User.findOrCreate(
      { where: { email: 'admin@test.com' } },
      { username: 'admin', email: 'admin@test.com', password: '123456' },
      callback);
  },
  adminRole: function(callback) {
    Role.findOrCreate(
      { where: { name: 'admin' } },
      { name: 'admin' },
      callback);
  }
}, function(err, results) {
  if (err) {
    return console.error(err);
  }
  console.log(results);
});
```

g. Indítsuk el az alkalmazást, látni kell a logban hogy létre lettek ezek hozva. Indítsuk el újra és látjuk majd hogy nem jöttek létre újak hanem az eddigiek frissültek.

h. Csináljuk meg a RoleMappinget

```
RoleMapping.findOrCreate(
  { where: { principalId: results.adminUser[0].id, roleId:
    results.adminRole[0].id } },
  {
    principalType: RoleMapping.USER,
    principalId: results.adminUser[0].id,
    roleId: results.adminRole[0].id
  }, function(err, principal) {
    if (err) {
      return console.error(err);
    }
    console.log('Principal:', principal);
  });
```

i. Indítsuk el az alkalmazást, nézzük meg a console-t.

j. Töröljük ki a fölösleges console.log parancsokat

3. Lépjünk be az adminnal

4. Próbáljuk meg lekérni a GET /users-t

a. 401 jön vissza

b. Hozzá kell adni az ACL-t a beépített User modellhez

5. Írjuk bele ezt a boot/auth.js-be

a. Snippet beillesztés

b. Property obj kitöltése:

```
acIs: [
  {
    accessType: '*',
    principalType: 'ROLE',
    principalId: 'admin',
    permission: 'ALLOW'
  }
]
```

c. Teszt, már le tudjuk kérni a GET /users-t

d. Próbáljunk meg usert is hozzáadni

Kliens generálás

1. Parancs: `lb-ng server\server.js client\angular-client.js`
2. Nézzük meg mi lett ebből
3. Generáljuk megadott modul névvel és url-el
 - a. `lb-ng server\server.js client\angular-client.js`
`-m chatService -u http://mychat-service.hu/api/v1/`

Statikus fájlok / kliens hostolása

1. client mappába rakjunk egy index.html-t
2. Nyissuk meg a localhost:3000-et
 - a. Nem az index.html nyílik meg hanem az api kezelő kapja el
3. server/boot/root.js-ben írjuk át a státusz urljét /status-re
4. Írjuk át a config.json-ben a base urlt /api/v1-re
5. Állítsuk be a static middleware-t a server/middleware.json-ban

```
"files": {  
  "loopback#static": {  
    "params": "$!../client"  
  }  
},
```

6. Teszt! <http://localhost:3000>

Build és deploy

1. Build:
 - a. Itt a `-n` belecsomagolja a függőségeket is
 - b. Nézzük meg a package.json-t utána
 - c. Nézzük meg mi generálódott