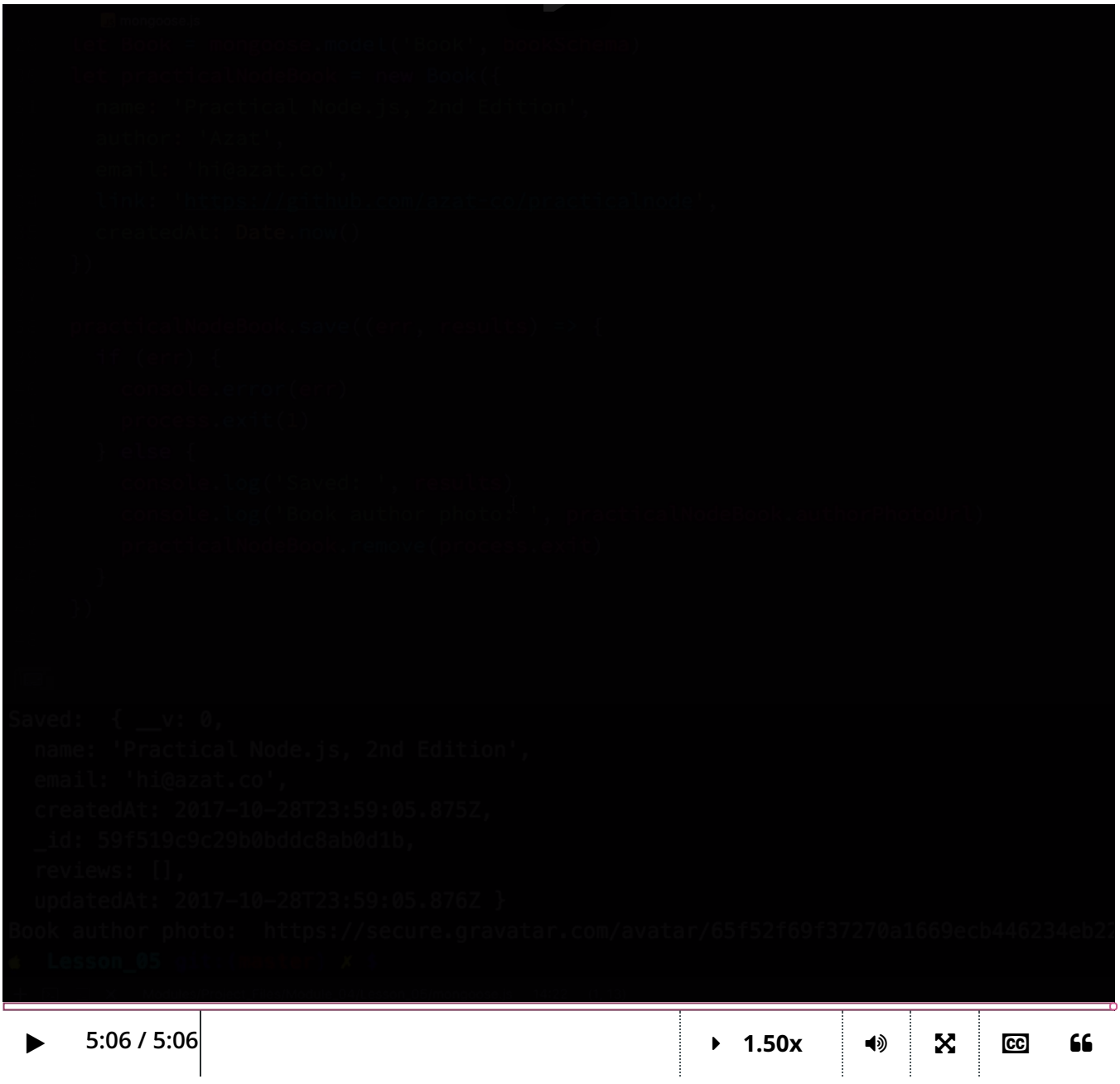
Virtual Fields
Video: Virtual Fields and Nested Documents



Video

Download video file

Transcripts

Download SubRip (.srt) file

Download Text (.txt) file

Virtual Fields

Virtuals are fields that don't exist in the database but act just like normal fields in a Mongoose document. To oversimplify, virtual fields are mock or fake fields that pretend to act and be normal ones.

Virtual fields are awesome for creating aggregate fields. For example, if our system requires to have first name, last name and the full name (which is just a concatenation of the first two names)—there's no need to store the full name values in addition to the first and last name values! All we need to do is concatenate the first and last name in a full name virtual.

Another use case is to make the database backward compatible. For example, we might have thousands of user items in a MongoDB collection and we want to start collecting their locations. We have two options: run a migration script to add the default location ("none") to the thousands of old user documents or use a virtual field and apply defaults at runtime!

To define a virtual we need to:

1. Call the `virtual(name)` method to create a virtual type (Mongoose API)
2. Apply a getter function with `get(fn)` (Mongoose API)

Gravatar, for example, is a service that hosts profile images. The URL is always an md5 hash of the user's e-mail. Therefore, we can get the virtual value (`gravatarUrl`) on the fly by hashing instead of storing the value (less overhead!).

In this example, we intentionally made the input email mixed cased and with a trailing space, and then applied `crypto:`

```
Identity.virtual('gravatarUrl')
  .get(function() {
    if (!this.email) return null
    var crypto = require('crypto'),
      email = "Hi@azat.co "
    email = email.trim()
    email = email.toLowerCase()
    var hash = crypto
      .createHash('md5')
      .update(email)
      .digest('hex')
    var gravatarBaseUrl = 'https://secure.gravatar.com/avatar/'
    return gravatarBaseUrl + hash
  })
```

Or, the case mentioned earlier—getting a full name out of first and last—is as follows:

```
userSchema.virtual('fullName')
  .get(function(){
    return this.firstName + ' ' + this.lastName
  })
```

Another scenario is when only a subset of the full document is exposed. For example, if the user model has tokens and passwords, we omit these sensitive fields by white-listing only the fields we want to expose:

```
userSchema.virtual('info')
  .get(function() {
    return {
      service: this.service,
      username: this.username,
      name: this.name,
      date: this.date,
      url: this.url,
      avatar: this.avatar
    }
  })
```

Virtual fields allow for a flexible way to create database fields without having to have the actual fields in the database.