Virtuals in Mongoose

Definition:

Virtuals are properties in Mongoose schemas that do not persist to MongoDB but are dynamically computed. They can be used to define getters for combining or formatting fields, and setters for decomposing values for storage.

Key Features

1. **Dynamic Properties**: Virtuals are computed on the fly and do not get saved to the database.
2. **Getters**: Useful for formatting or combining fields.
3. **Setters**: Allow you to decompose a single value into multiple fields for storage.

Example: Defining a Schema with Virtuals

```javascript
const mongoose = require('mongoose');
const { Schema } = mongoose;

const personSchema = new Schema({
  name: {
    first: String,
    last: String
```

  }
});

const Person = mongoose.model('Person', personSchema);

// Create a document
const axl = new Person({
  name: { first: 'Axl', last: 'Rose' }
});

console.log(axl.name.first + ' ' + axl.name.last); // Axl Rose
~~~
Instead of manually concatenating `first` and `last` every time, you can use a virtual.

---

### Adding a Virtual Getter

#### Option 1: Adding to Schema Options
```javascript
const personSchema = new Schema({
  name: {
    first: String,
    last: String
  }
}, {
  virtuals: {
~~~

```
    fullName: {
      get() {
        return this.name.first + ' ' + this.name.last;
      }
    }
  }
});
```

Option 2: Using the `virtual` Method

```javascript
personSchema.virtual('fullName').get(function() {
  return this.name.first + ' ' + this.name.last;
});
```

Now, accessing the `fullName` property dynamically computes the full name:

```javascript
console.log(axl.fullName); // Axl Rose
```

Including Virtuals in `toJSON()` and `toObject()`

By default, virtuals are not included when converting a document to JSON or a plain JavaScript object.

Explicit Inclusion:

```javascript
const obj = axl.toObject({ virtuals: true });
console.log(obj.fullName); // Axl Rose
```

Automatic Inclusion:

```javascript
const personSchema = new Schema({
  name: {
    first: String,
    last: String
  }
}, {
  toJSON: { virtuals: true } // Automatically include virtuals
});
```

For `JSON.stringify()`:

```javascript
console.log(JSON.stringify(axl.toObject({ virtuals: true }))); // Includes `fullName`
```

Adding a Setter

You can use virtuals to set both `first` and `last` names from a single `fullName` string.

Option 1: Adding to Schema Options

```javascript
const personSchema = new Schema({
  name: {
    first: String,
    last: String
  }
}, {
  virtuals: {
    fullName: {
      get() {
        return this.name.first + ' ' + this.name.last;
      },
      set(v) {
        this.name.first = v.substr(0, v.indexOf(' '));
        this.name.last = v.substr(v.indexOf(' ') + 1);
      }
    }
  }
});
```

Option 2: Using the `virtual` Method

```javascript
personSchema.virtual('fullName')
  .get(function() {
    return this.name.first + ' ' + this.name.last;
  })
  .set(function(v) {
    this.name.first = v.substr(0, v.indexOf(' '));
    this.name.last = v.substr(v.indexOf(' ') + 1);
  });

// Example usage
axl.fullName = 'William Rose';
console.log(axl.name.first); // William
console.log(axl.name.last);  // Rose
```

Caveats

1. **Querying**: Virtuals cannot be used in MongoDB queries since they are not stored in the database.
   ```javascript
   // This won't work
   Person.find({ fullName: 'Axl Rose' });
   ```

2. **Validation**: Virtual setters are applied before other validations.

Summary

Virtuals provide a powerful way to dynamically compute properties and manipulate values. They improve code readability and modularity, making it easier to work with formatted or combined data without altering the database structure.