

# Mongoose Methods Use Cases

Mongoose method what are they used for most commonly

In **Mongoose**, **methods** are typically used to define **custom instance methods** on documents. That means they are used to add **custom behavior** to individual MongoDB documents you retrieve or create using a Mongoose model.


## Most Common Use Cases for Mongoose Methods

### 1. Password Verification

Used to compare a plaintext password with a hashed password (e.g. with bcrypt):

```
js

userSchema.methods.isPasswordCorrect = async function (password) {
  return await bcrypt.compare(password, this.password);
};
```

-  `this.password` refers to the document's password.
- Useful during **login** to verify password matches.

### 2. JWT Token Generation

Generate authentication tokens from a user instance:

```
js

userSchema.methods.generateJWT = function () {
  return jwt.sign(
    { id: this._id, username: this.username },
    process.env.JWT_SECRET,
    { expiresIn: "1d" }
  );
};
```

### 3. Custom Business Logic

For example, a method to get the user's full name:

```
js
```

```
userSchema.methods.getFullName = function () {  
  return `${this.firstName} ${this.lastName}`;  
};
```

#### 4. Data Formatting / Virtual Calculations

Example: Calculating age from a date of birth.

```
js
```

```
userSchema.methods.getAge = function () {  
  const today = new Date();  
  const birthDate = new Date(this.dob);  
  let age = today.getFullYear() - birthDate.getFullYear();  
  const m = today.getMonth() - birthDate.getMonth();  
  if (m < 0 || (m === 0 && today.getDate() < birthDate.getDate())) {  
    age--;  
  }  
  return age;  
};
```

#### When to Use `.methods` vs `.statics`

Feature	<code>.methods</code> (instance)	<code>.statics</code> (model-level)
Called on	A document instance (e.g. <code>user</code> )	A model (e.g. <code>User</code> )
Access to <code>this</code>	Refers to the document	Refers to the model
Use case	Password check, token generation, formatting	Custom queries, batch operations, etc.

#### Example Usage:

js

```
const user = await User.findOne({ username: "ankit" });
if (await user.isPasswordCorrect("secret123")) {
  const token = user.generateJWT();
  // login success
}
```


Let me know if you want the difference with **virtuals**, **statics**, or **plugins** too.

is it used for queries?

Great question — **Mongoose** `.methods` are *not* used for queries.

Let's break this down:

### Are `.methods` used for queries?

 No, `.methods` are used on individual documents, *not* for querying the database.

They are for:

- Processing or modifying **one document**
- Performing checks or transformations **after** a document is fetched (like password checking)

### What is used for queries then?

You use:

1. **Model functions** like:

js

```
User.find(), User.findOne(), User.findById()
```

## 2. Custom `.statics` (Mongoose static methods):

These are the ones you define for **custom queries**.

Example:

```
js



userSchema.statics.findByUsername = function (username) {
  return this.findOne({ username });
};
```

Usage:

```
js

const user = await User.findByUsername("ankit");
```

## Summary

Feature	Used for Queries?	Called on	Example Use
<code>.methods</code>	 No	Document instance	<code>user.isPasswordCorrect()</code>
<code>.statics</code>	 Yes	Model	<code>User.findByUsername()</code>

If you're building **login or authentication**, you'd often:

- Use **statics** to find a user by email/username
- Then use **methods** on that document to verify the password

Let me know if you'd like a full login/auth example using both `statics` and `methods`.





okay tell me the use of methods then like where it used method creation and it;s example

Perfect! Let's go deep into **Mongoose** `.methods` — where and how they're used — with a **real-world example**.

## What are Mongoose `.methods`?

- `.methods` is used to define **custom instance methods** on **documents**.
- After you fetch a document from MongoDB, you can call these methods directly on it.
- They help encapsulate behavior related to a **single document**.

## Common Use Cases for `.methods`

Use Case	Purpose
 Password verification	Compare plain password with hashed one
 JWT token generation	Generate auth token from user data
 Derived values	Compute values (like full name, age, etc.)
 Document formatting	Format response data or remove sensitive fields

## Example: User Model with `.methods`

```
js
```

```

import mongoose from "mongoose";
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";

const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
  },
  password: {
    type: String,
    required: true,
  },
});

// 🗝️ 1. Password Verification Method
userSchema.methods.isPasswordCorrect = async function (password) {
  return await bcrypt.compare(password, this.password);
};

// 🔑 2. JWT Generation Method
userSchema.methods.generateJWT = function () {
  return jwt.sign(
    { id: this._id, username: this.username },
    process.env.JWT_SECRET,
    { expiresIn: "1d" }
  );
};

// 🧠 3. Get Username in Uppercase
userSchema.methods.getDisplayName = function () {
  return this.username.toUpperCase();
};

const User = mongoose.model("User", userSchema);
export default User;

```

## → 📱 Where Do You Use `.methods`?

After querying the DB and getting a user **document**, like this:

```
js

const user = await User.findOne({ username: "ankit" });

if (!user) {
  throw new Error("User not found");
}

// ✅ Call your methods like this:
const isMatch = await user.isPasswordCorrect("mypassword");
const token = user.generateJWT();
const displayName = user.getDisplayName();

console.log({ isMatch, token, displayName });
```

## 💡 Why Use `.methods`?

- Keeps logic **inside the model**, not scattered across routes or services.
- Encourages **OOP style** (each user document “knows” how to check its password, generate a token, etc.)
- Easy to test, reuse, and maintain.

Let me know if you want a full example with `.statics` too or want to combine it into an authentication flow!

```
const user = await User.findOne({ username: "ankit" });
```

so what is actually happening when i receive the output from the query?  
what it returns which allows me to use methods?

Excellent question. Understanding **what's returned from Mongoose queries** and **how `.methods` work on it** gives you a real grip on how Mongoose operates under the hood.

## What happens in this line?

js

```
const user = await User.findOne({ username: "ankit" });
```

Here's what's happening step-by-step:

### ✓ 1. `User.findOne(...)` returns a Mongoose Document

- The result is not a plain JavaScript object.
- It's an **instance of your Mongoose model**, like an object created with `new User(...)`.

js

```
user instanceof mongoose.Model; // ✓ true
```

- That instance **inherits** the `.methods` you defined on the schema.

## What allows `user.isPasswordCorrect()` to work?

When you defined this in your schema:

js

```
userSchema.methods.isPasswordCorrect = async function (password) { ... }
```

...it was added to the **prototype** of the model's documents. So when you run:



```
js
const user = await User.findOne(...);
await user.isPasswordCorrect("abc123");
```

You're calling a method on that document instance because:

- Mongoose uses the schema to **construct a class** (think of it like `class UserDocument extends mongoose.Document`)
- Your `user` is an instance of that class
- `.methods` get attached to the prototype of that class

So `user.isPasswordCorrect()` works because it's part of the prototype chain.



## What does Mongoose actually return?

Mongoose returns an object that looks like this:

```
js
{
  _id: ObjectId("..."),
  username: "ankit",
  password: "hashed_password",
  __v: 0,
  // plus any custom methods you've added
  isPasswordCorrect: [Function],
  generateJWT: [Function],
}
```

But it's more than a plain object — it's a **fully-powered Mongoose Document** with:

- Mongoose metadata
- Built-in functions (like `.save()`, `.toObject()`, etc.)
- Your custom `.methods`



## Recap:

Concept	Explanation
<code>User.findOne(...)</code>	Returns a <b>Mongoose Document instance</b>
Schema <code>.methods</code>	Get attached to each <b>document's prototype</b>
Resulting object	Has both data ( <code>_id</code> , fields) and functions (your methods)
Why <code>.isPasswordCorrect()</code> works	Because the result is <b>not a plain object</b> , but a <b>document instance</b>

Let me know if you want to peek into `Object.getPrototypeOf(user)` to see all the functions it inherits too.