```
{
    "success": false,
    "message": "next is not a function"
} why tthis type of error occur
```

This error usually comes from **JavaScript** and means:

> You're trying to call `next()` but **the thing called** `next` **is not actually a function**.

Common reasons this happens

1. `next` **was never passed in**
This often happens in **Express.js middleware**.

❌ Example:

```
js
Copy code

app.use((req, res) => {
  next(); // ❌ next is undefined
});
```

✅ Fix:

```
js

app.use((req, res, next) => {
  next(); // ✅ now it's a function
});
```

👉 Middleware functions **must have** `next` **as the third parameter**.

---

2. You overwrote `next` **with something else**
You might have accidentally reassigned it.

❌ Example:

```
js
Copy code

function middleware(req, res, next) {
  next = true;    // ❌ overwriting next
  next();         // error
}
```

✅ Fix:
Never reassign `next`.

---

3. Using `next()` **outside middleware**
`next()` only exists in middleware or route handlers.

❌ Example:

```
js
Copy code

function helper() {
  next(); // ❌ next does not exist here
}
```

✅ Fix:
Pass `next` as an argument:

```
js
Copy code

function helper(next) {
  next();
}
```

---

4. Wrong framework or context
If you're not using **Express / Koa / similar middleware-based frameworks**, `next()` may not exist at all.

---

5. Promise / async confusion
Calling `next()` inside async code without proper middleware structure can also trigger this.

How to debug quickly

Add this line before calling `next()`:

```
js
Copy code

console.log(typeof next);
```

- If it prints `"function"` → OK
- If it prints `"undefined"` or `"object"` → that's the problem

In short

This error occurs because:

> **JavaScript expected** `next` **to be a function, but it wasn't.**

If you want, you can paste the **exact code** where the error happens and I'll point out the issue precisely 👍

```
userSchema.pre("save", async function(next) {
  if (!this.isModified("password")) return next();
  try {
    this.password = await bcrypt.hash(this.password, 10);
    next();
  } catch (err) {
    next(err);
  }
});
```

Your code is **almost correct**, but this error happens because of a **Mongoose hook rule**:

> ❌ **You should not mix** `async/await` **with** `next()` **in Mongoose middleware**

That's why you're getting `next is not a function`.

Why this happens

In Mongoose:

- If you **use** `next`, the function should **NOT be async**
- If you **use** `async`, Mongoose **automatically handles** `next`

Right now, you're doing **both**, which causes Mongoose to mis-handle `next`.

❌ Your current code (problematic)

```
js
Copy code

userSchema.pre("save", async function(next) {
    if (!this.isModified("password")) return next();
    try {
        this.password = await bcrypt.hash(this.password, 10);
        next();
    } catch (err) {
        next(err);
    }
});
```

✅ Fix Option 1 (Recommended): Use `async` WITHOUT `next`

```
js
Copy code

userSchema.pre("save", async function () {
    if (!this.isModified("password")) return;

    this.password = await bcrypt.hash(this.password, 10);
});
```

- Clean
- Modern
- No `next()` needed
- Mongoose waits automatically

✅ Fix Option 2: Use `next` WITHOUT `async`

```
js
Copy code

userSchema.pre("save", function (next) {
    if (!this.isModified("password")) return next();

    bcrypt.hash(this.password, 10)
        .then(hash => {
            this.password = hash;
            next();
        })
        .catch(err => next(err));
});
```

✅ Best Practice (Use Option 1)

Mongoose v5+ **prefers async middleware**.
So your best version is:

```
js
Copy code

userSchema.pre("save", async function () {
    if (!this.isModified("password")) return;
    this.password = await bcrypt.hash(this.password, 10);
});
```

Summary

The error occurs because:

- `async` + `next()` were used together
- Mongoose expects **one or the other**, not both

If you want, tell me:

- Your **Mongoose version**
- Where this schema is used

and I'll double-check everything 👍