You are absolutely correct! You spotted a hidden detail that most people miss.

The "hashed password" string you see in your database (e.g., `$2b$12$HKL...`) is actually a **Package**. It contains **three** separate pieces of information packed into one string.

If it didn't contain the salt, the `verify()` function would never work!

The Anatomy of a Bcrypt Hash

Let's dissect a real Bcrypt string:

`$2b$12$EixZaYVK1fsz1jd2...`

It is split by the dollar signs `$`.

1. **The Algorithm (`$2b`):**

- This tells the system: *"I was created using the **Bcrypt** algorithm (version 2b)."*
- This is how `verify()` knows which math formula to use.

2. **The Work Factor (`$12`):**

- This is the "Cost" or "Strength".
- It tells the system: *"I was hashed 2^12 (4096) times."*
- This ensures `verify()` does the exact same amount of work to match the hash.

3. **The Salt (The First 22 characters of the last part):**

- This is the random data added to the password.
- **Crucial:** The system *stored* the random salt right here in the string!

4. **The Actual Hash (The rest of the string):**

- The final mathematical result.

How `verify()` uses this info

When you run `verify("secret123", "$2b$12$salt...hash...")`:

1. **It reads the string:** It extracts the salt from the middle of the stored string.
2. **It does the math:** It takes your input (`"secret123"`) + the extracted salt.
3. **It runs the algorithm:** It hashes them together.
4. **It compares:** It checks if the *new* result matches the *last part* of the stored string.

So yes, the "hashed password" is actually a self-contained instruction manual on how to verify itself.

Ready for the next file? Please paste **`routers/user.py`**.