Note: Black Box Testing and White Box Testing will be focused on the Account Creation, in particular the Register Account Use Case

1. Black Box Testing

I. AuthController (AuthRoutes)

Class to Test: auth_routes.py (Flask Blueprint)

The AuthRoutes Flask Blueprint handles user authentication workflows, primarily focusing on:

a. Registration:
   - Creates seller/agent accounts with validation for phone numbers (+65 format), password strength (8+ chars with mixed case, numbers, and special characters), and unique usernames.
   - For agents, verifies CEA license IDs against a government API from data.gov.sg
   - Link: https://data.gov.sg/datasets/d_07c63be0f37e6e59c07a4ddc2fd87fcb/view?dataExplorerPage=4

b. OTP Verification:
   - Integrates with Twilio to send SMS OTPs (or mocks for testing) and validates user-submitted codes.
   - Currently only works for one phone number due to API free trial account limitations

c. Session Management:
   - Temporarily stores OTPs in session and clears them post-verification.

d. Data Persistence:
   - Saves validated user data to SQLite (accounts.db) with role-based fields (e.g., agent_id).

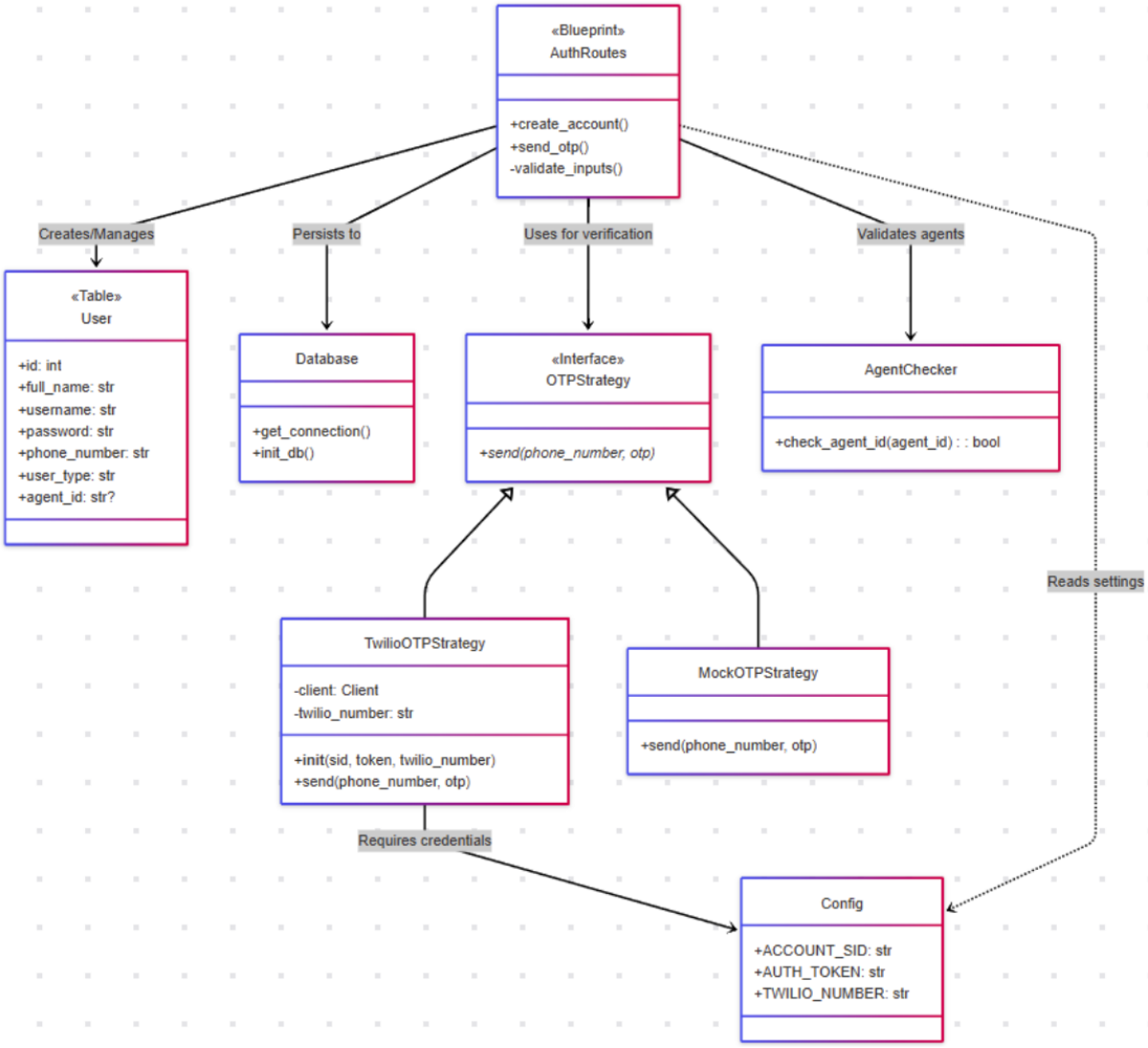
Key Features:

- Auto-formats Singaporean phone numbers (e.g., 91234567 → +6591234567).
- Uses the Strategy Pattern for OTP services (refer to images below or refer to 2006-SCSB-T4/HomeProperty/services/otp_service.py)
- Enforces strict input validation before database operations.

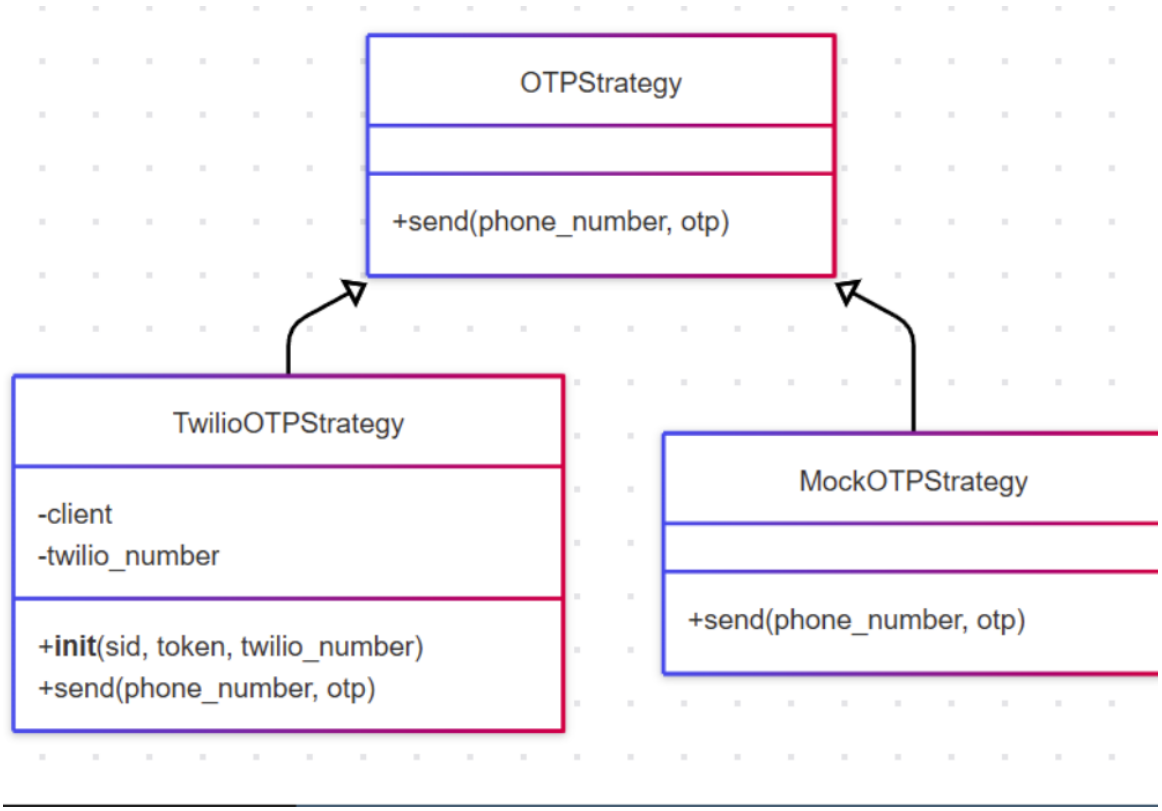
Example Flow:
User submits form → OTP sent → Validates inputs → Creates account → Redirects

Class Diagram:

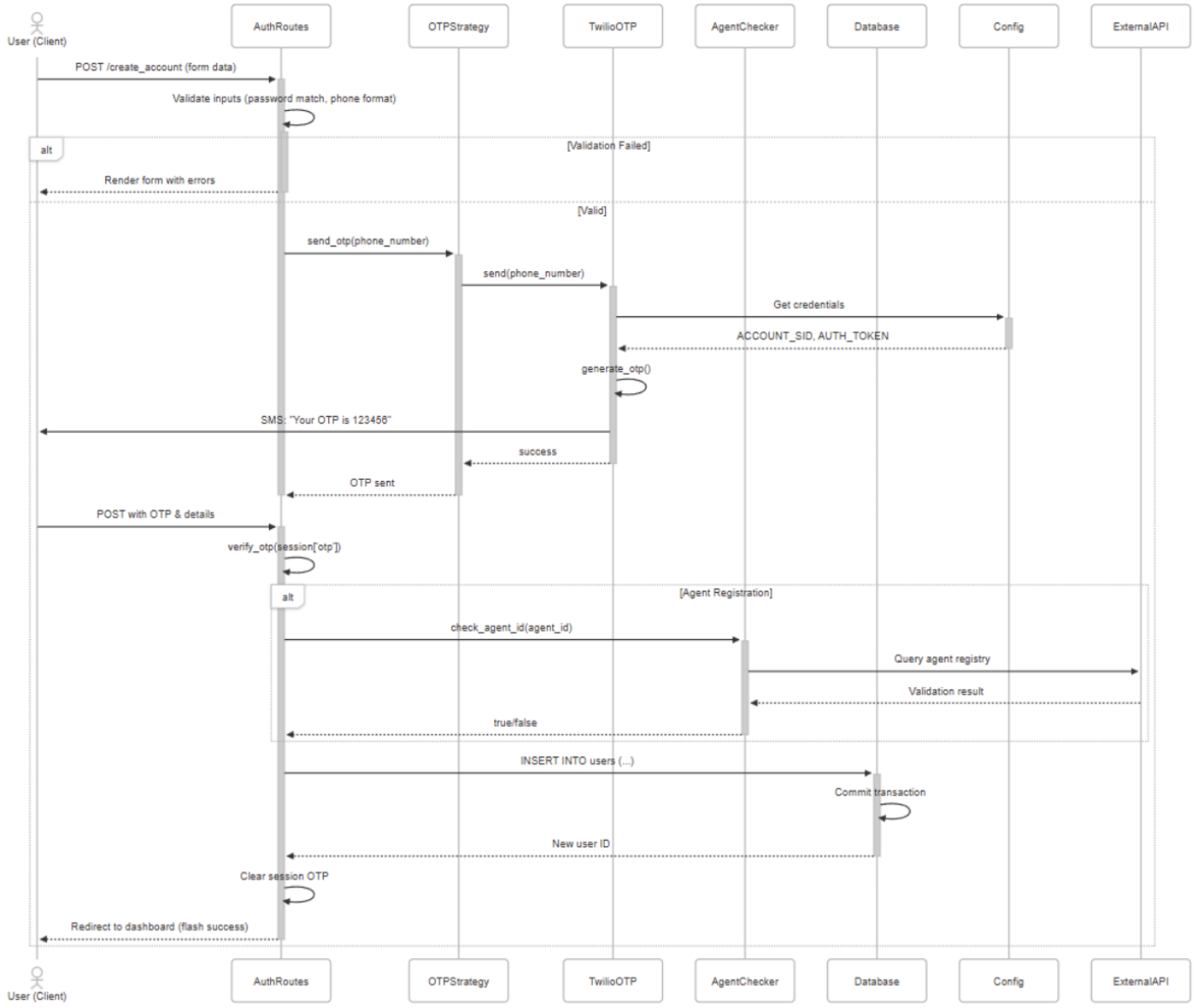
Refer to Lab 4 Folder if image is unclear



OTP Strategy (included in Class Diagram):



Sequence diagram:
Refer to Lab 4 Folder if image is unclear



Equivalence Class and Boundary Value Testing (Under Black Box Testing)

1. Registration Function

   Valid Equivalence Class:
   - Full name, username, password, phone number, and user type with correct formats
   - Password: 8+ characters with uppercase, lowercase, number, and special character
   - Phone number: 8 digits with or without +65 prefix (system auto-formats)
   - OTP: 6-digit code matching sent value

   Invalid Equivalence Class:

   - Missing required fields (name, username, password, etc.)
   - Passwords with incorrect formats (missing rules)
   - Invalid phone numbers (≠8 digits, wrong prefix)
   - Incorrect/expired OTP codes
   - If role selected is Seller:

   Valid: No agent ID required

   Invalid: Agent ID field filled (should be stored as NULL)

   - If role selected is Agent:

   Valid: CEA agent ID registered in government database

   Invalid: Invalid/empty agent ID

2. Boundary Value Testing

   - Phone Number:
     a. Lower bound: 7 digits (9123456) → Invalid
     b. Upper bound: 8 digits (91234567) → Valid

   - Password Length:
     a. Lower bound: 7 characters (Apple1!) → Invalid
     b. Upper bound: 8 characters (Apple1!@) → Valid

   - OTP Code:
     a. Lower bound: 100000 → Valid if correct
     b. Upper bound: 999999 → Valid if correct

Test Case #	Test Input	Expected Output	Actual Output (Tick means same as Expected Output)
1	Valid seller account (unique username, strong password, verified phone)	Account created successfully, redirect to login	✓
2	Valid agent account (valid agent ID, unique username, strong password, verified phone)	Account created successfully, redirect to login	✓
3	Duplicate username (no 2 accounts can have same username regardless of account type)	Error: "Username already taken"	✓
4	Password without uppercase letter	Error: "Password must contain uppercase letter"	✓
5	Password without lowercase letter	Error: "Password must contain lowercase letter"	✓
6	Password without number	Error: "Password must contain number"	✓
7	Password without special character	Error: "Password must contain special character"	✓
8	Phone number without +65 prefix	System auto-prepends +65, OTP sent	✓
9	Invalid OTP entered	Error: "Invalid OTP"	✓
10	Agent account with invalid agent ID	Error: "Invalid agent registration number"	✓
11	Agent account with empty agent ID	Error: "Agent registration number required"	✓
12	Missing required field (e.g. empty username)	Error highlighting missing field	✓
13	Password and confirm password don't match	Error: "Passwords do not match"	✓
14	Seller account with agent ID field filled	Agent ID stored as NULL in database	✓
15	Multiple OTP requests for same number	Only latest OTP is valid	✓
16	Valid credentials but Twilio SMS failure	Error: "Failed to send OTP" with retry option	✓
17	Session timeout during OTP verification	Prompt to restart verification process	✓
18	Existing user forgot password flow	Successfully resets password after OTP verification	✓
19	Password with 7 characters (just below min)	Error: "Password must be at least 8 characters long..."	✓

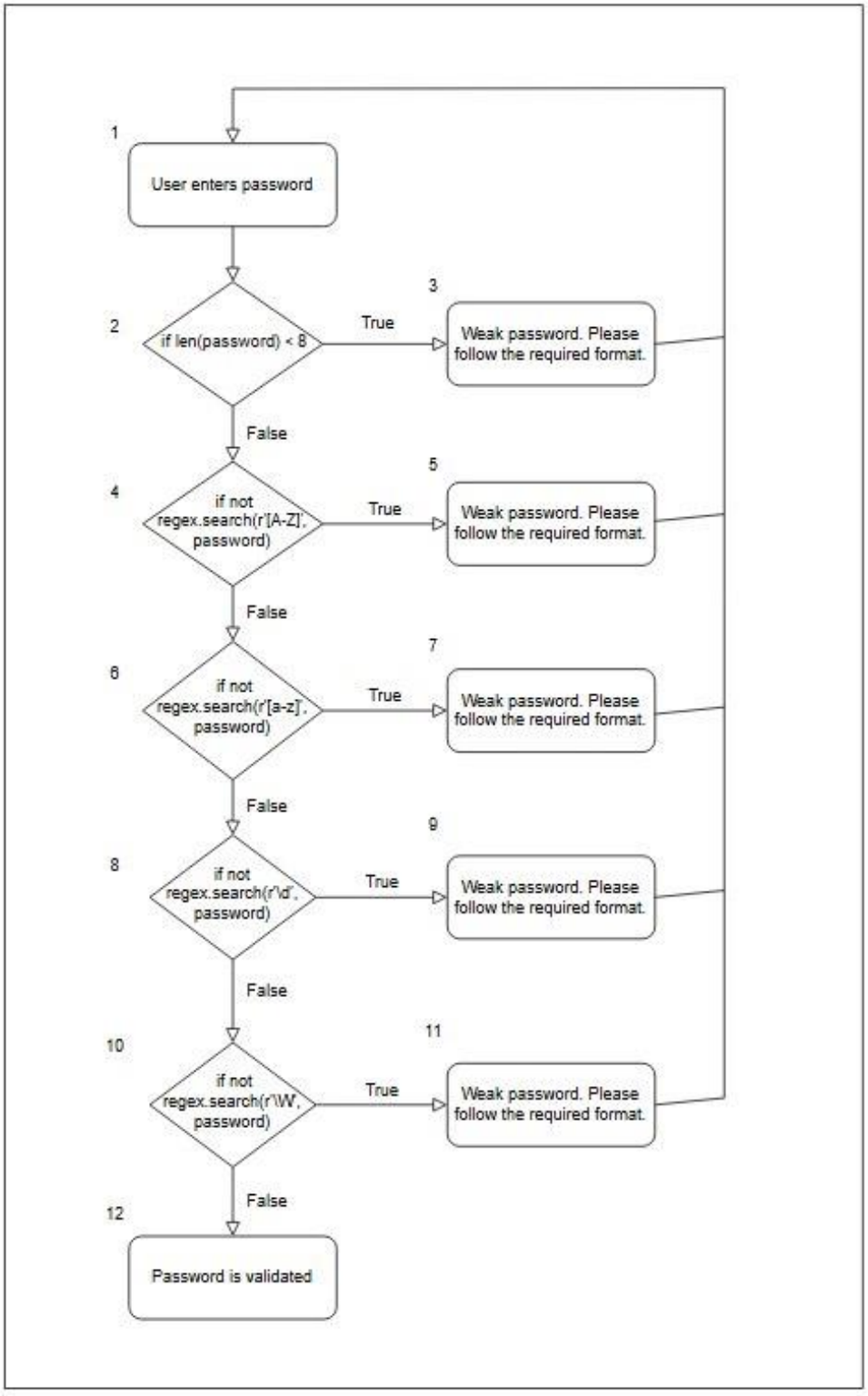
20	Password with exactly 8 characters (min valid)	Account created successfully	✓
21	OTP = 100000 (lower boundary)	OTP accepted if correct	✓
22	OTP = 999999 (upper boundary)	OTP accepted if correct	✓
23	Username with exactly max allowed length (e.g., 20 characters)	Account created successfully	✓
24	Username exceeding max allowed length (e.g., 21 characters)	Error: "Username too long"	✓

2. White Box Testing

1. Strong Password Check

```python
def is_strong_password(password):
    """Check if the password meets strength requirements."""
    if len(password) < 8:
        return False
    if not regex.search(r'[A-Z]', password):  # Uppercase letter
        return False
    if not regex.search(r'[a-z]', password):  # Lowercase letter
        return False
    if not regex.search(r'\d', password):     # Digit
        return False
    if not regex.search(r'\W', password):     # Special character
        return False
    return True
```

1.1 Control Flow Graph



1.2 Cyclomatic Complexity

Cyclomatic Complexity (CC) is calculated as the number of binary decision points + 1.
From the control flow diagram, there are five decision points:
1. Length check
2. Uppercase letter check
3. Lowercase letter check
4. Digit check
5. Special character check

Thus,
Cyclomatic Complexity (CC) = 5 + 1 = 6

1.3 Basis Paths

The following are the independent basis paths derived from the control flow graph:

- Path #1: Password length is less than 8 → return False
- Path #2: Password length valid, but missing uppercase letter → return False
- Path #3: Password length and uppercase present, but missing lowercase → return False
- Path #4: Password meets all except digit → return False
- Path #5: Password has all but no special character → return False
- Path #6 (Baseline): Password meets all conditions → return True

1.4 Test Cases and Results

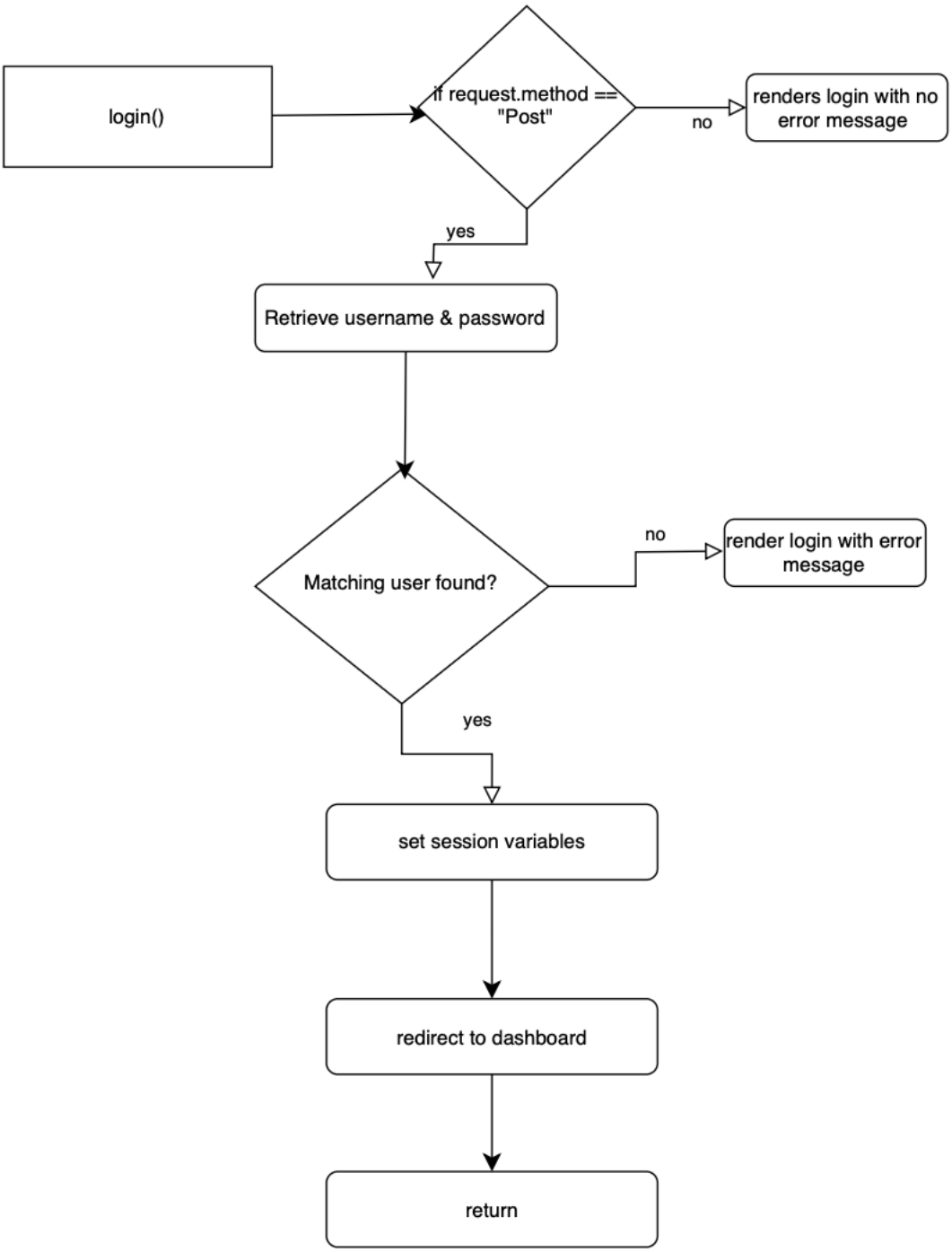
No.	Test Input	Expected Output	Actual Output	Pass?
1	Password : Short1! (Too short, 7 chars)	Weak password. Please follow the required format.	Weak password. Please follow the required format.	Yes
2	Password : longpassword1! (No uppercase)	Weak password. Please follow the required format.	Weak password. Please follow the required format.	Yes
3	Password : LONGPASSWORD1!(No lowercase)	Weak password. Please follow the required format.	Weak password. Please follow the required format.	Yes

4	Password : StrongPass! (No digit)	Weak password. Please follow the required format.	Weak password. Please follow the required format.	Yes
5	Password : StrongPass1 (No special char)	Weak password. Please follow the required format.	Weak password. Please follow the required format.	Yes
6	Password : StrongPass1! (Meets all requirements)	Successful login	Successful login	Yes

2. Login

```
def login():
    if request.method == 'POST':
        username = request.form['username'].strip()
        password = request.form['password'].strip()
        user_type = request.form['user_type'].strip()

        conn = sqlite3.connect('accounts.db')
        c = conn.cursor()
        c.execute("SELECT * FROM users WHERE username=? AND password=? AND user_type=?",
                  (username, password, user_type))
        user = c.fetchone()
        conn.close()

        if user:
            session['username'] = username
            session['user_type'] = user_type
            session['full_name'] = user[1]
            return redirect('/seller_dashboard' if user_type == 'seller' else '/agent_dashboard')

        return render_template('login.html', error_message="Invalid credentials or account does not exist.")

    return render_template('login.html')
```

2.1 Control Flow Graph



2.2 Cyclomatic Complexity

There are 2 decision points:

1. if request.method == 'POST'
2. if user: (i.e., user found in DB)

Thus,
Cyclomatic Complexity (CC) = 2 + 1 = 3

2.3 Basis Paths

• Path #1: POST request with valid credentials (user in DB) → redirect to seller or agent dashboard respectively
• Path #2: POST request with invalid credentials (user not found) → render login page with error message
• Path #3: non POST request → render login page with no error (blank page with inputs yet to be filled)

2.4 Test Cases and Results

No.	Test Input	Expected Output	Actual Output	Pass?
1	"Username: Larry20, Password: Password21@, Account Type: seller" (in DB)	Redirect to / seller dashboard	Redirect to / seller dashboard	Yes
2	"Username: alice, Password: Alice123!, Account Type: seller" (not in DB)	"Invalid credentials or account does not exist"	"Invalid credentials or account does not exist"	Yes
3	"Username: Larry20, Account Type: Property Agent"	"Fill out this field"	"Fill out this field"	Yes