Manual Testing Defect Log

Project Name: User Registration and Log in

Test Method: Manual Testing (Black Box & White Box Testing)

Tester:

SWE2309509 He Bolin

SWE2309520 Li Ruibing

SWE2309540 Wu Yi

Section 1: Usability

Defect ID	Test case	Title	Module	Severity	Steps	Expected Results	Actual Results	Suggestion
Defect_01	REG_4.7 / Static check	Password rule inline hints are not displayed during	Register – Password	Medium	1. Navigate to the Register page. Click (focus) the	Password rule hints should appear	No rule hints are displayed; no	Add real-time password rule hints that appear

	(White Box Testing)	registration	Field		Password field 2. Type password characters	immediately.	real-time feedback is provided.	when the user focuses on or types in the password field.
Defect_02	REG_6.2	Username length is not validated in real-time.	Register – Username Field	Medium	Enter 51-char username	Immediate inline error when limit exceeded.	Error is shown only after submitting the entire form.	implement real-time username length validation and show an immediate inline error when the limit is exceeded.
Defect_03	REG_2.5	Invalid inputs are not fully validated	Register – Input Validation	High	1. Enter invalid inputs in all fields 2. Click Register	All the invalid inputs should be highlighted	Only 'Invalid Email' is highlighted	Add full field-level validation, so every invalid input is highlighted with a clear error message.

Defect_04	REG_2.6 /Static check (White Box Testing)	Verify password mask of input password and confirm password	Register – Input Validation	Medium	1. Enter valid inputs in all fields 2. Try to unmask the password and confirm the password	The input password is masked by default; the user can click on the 'Show/Hide password' button to show or hide the password.	The system has no design for 'Hide/Show password'.	Add a standard Show/Hide password toggle for both password and confirm password fields.
Defect_05	Static check (White Box Testing)	Confirm Password does not revalidate when Password changes	Register – Validation	Low	1. Register form using **Zod + React Hook Form** 2. Enter password = 123. 3. Enter confirm Password = 123. (valid) 4. Change password to 123456. 5. Observe confirm Password field	Confirm Password should immediately show mismatch error after password changes.	Confirm Password remains valid until user interacts with confirmed field or submits form.	Trigger revalidation of the Confirm Password field whenever the Password value changes, so mismatch errors are shown immediately.

					validation state.			

Section 2: Security

Defect ID	Test case	Title	Module	Severity	Steps	Expected Results	Actual Results	Suggestion
Defect_06	LGN_6.3	Account lockout does not persist across different IP addresses	Login – Security / Lockout Policy	High	1. Enter an invalid username and password 2. In one IP, click login	In the other IP, the system should still be locked and not	Further login attempt is allowed in a new IP (raises security problems)	Enforce lockout status at the account level rather than IP

					until the system is locked 3. Switch to another IP and try to log in with same username and password	allow further login attempts		level, so locked accounts cannot log in from any IP.
Defect_07	LGN_6.4	Account lockout can be bypassed by entering a valid password after multiple failed attempts	Login – Security / Lockout Policy	High	1. Enter invalid credentials 4 times (failure counter increments). 2. Enter a valid username/password once (login succeeds). 3. Logout. 4. Enter invalid credentials several times.	The system should lock the account once the total number of failures reaches 5, regardless of successful logins in between.	After a valid login, the failure count resets. Users can attempt an invalid login up to five more times. The system does not lock until 5 new failures occur. This bypasses the lockout policy.	Ensure that once the account reaches the failure limit, it remains locked regardless of successful login attempts until the lockout duration or admin reset is applied.
Defect_08	DAT_1.4	User session remains valid after	Dashboard – Authenticati	High	1. Login with a valid registered account	Once the user account no	1. Database shows user `user01` no	Immediately invalidate the

		user is deleted from the database	on / Session Management		2. Enter dashboard page 3. Check database 4. Delete user data and observe interface 5. Check database 6. Refresh the page for multiple times, observe the interface	longer exists in the database, the system should: •Immediately invalidate the session token • Log the user out • Redirect to Login Page • Prevent access to the dashboard or other protected routes	longer exists. However, the session remains valid. 2. Users are still able to stay on the dashboard even after multiple refreshments. 3.System does not redirect or invalidate tokens, causing a security risk.	active session token when the user record is removed and force logout on the next request.
Defect_09	DAT_1.7	Old session token remains valid after password modification	Login – Authentication / Token Management	High	1. Register an account with password: `Abc123!@`. 2. Manually update the password in the database to:	- Login with new password should succeed. - Login with old password should		Invalidate all existing tokens whenever a password is updated, so old tokens cannot

					`123Abc!@`. 3. Attempt login with new password. 4. Attempt login with old password. 5. Attempt login using previously issued token (before password modification).	fail. - Any previously issued token should become invalid immediately after password modification. - Access to old tokens should be rejected.		be used to authenticate.
Defect_10	SEC_1.6	JWT access token stored directly in local Storage	Frontend – Authentication / Token Handling	High	1. Login using a valid account. 2. Open DevTools → Application → Local Storage. 3. Inspect values stored under the app domain.	No sensitive tokens should be stored in Local Storage. Tokens should be stored using secure HttpOnly cookies to prevent XSS	The raw JWT access token is clearly visible in Local Storage. Token is accessible to any injected JavaScript, creating a severe XSS exploitation risk.	Store JWT tokens in secure HttpOnly cookies instead of Local Storage to prevent XSS token theft.

						leakage.		
Defect_11	SEC_1.7	Backend uses hardcoded fallback SECRET_KEY when environment variable is missing	Backend – Security / Configuration	High	1. Manually analyze backend authentication configuration. 2. Inspect SECRET_KEY initialization logic.	The active SECRET_KEY should be random or loaded from environment variables. It should not match the known default key.	The code explicitly falls back to a hardcoded default value:"0**9d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7**", which raises security problem	Remove all hardcoded SECRET_KEY defaults and require a valid environment variable; fail startup if SECRET_KEY is missing.
Defect_12	Static check (White Box Testing)	Token stored prematurely before user info retrieval, causing inconsistent session state	Login – Authentication	High	1. Perform static code review on **LoginPage.tsx.** 2. Locate **loginMutation.onSuccess(...).** 3. Observe that **setToken(token)** executes before	Tokens should only be stored after user information is successfully fetched. If user info fetch fails, token should be cleared or	Token is already written to **AuthStore** even when **getCurrentUser()** fails. This leads to a **"half-logged-in"** inconsistent	Store the token only after user information loads successfully; if fetching user info fails, clear the token and prevent partial

					awaiting **authApi.getCurrentUser().** 4. Inspect the catch block and note missing token rollback logic.	should never be stored.	authentication state.	login state.
Defect_13	Static check (White Box Testing)	**isLockedOut** does not persist across refresh and gives misleading security feedback	Login – Security / UI	Medium	1. Inspect lockout logic in **onError** handling status **429**. 2. Observe that lockout state is stored only in component memory. 3. Recognize that page to refresh resets throughout the entire state.	Lockout state should persist (via backend-provided timestamp or session storage), preventing misleading access attempts.	After refreshing, lockout appears cleared even though backend still blocks the login.	Persist lockout status using backend data or storage (e.g., timestamp or token), so the UI reflects the correct lockout state after refreshing.

Section 3: Reliability

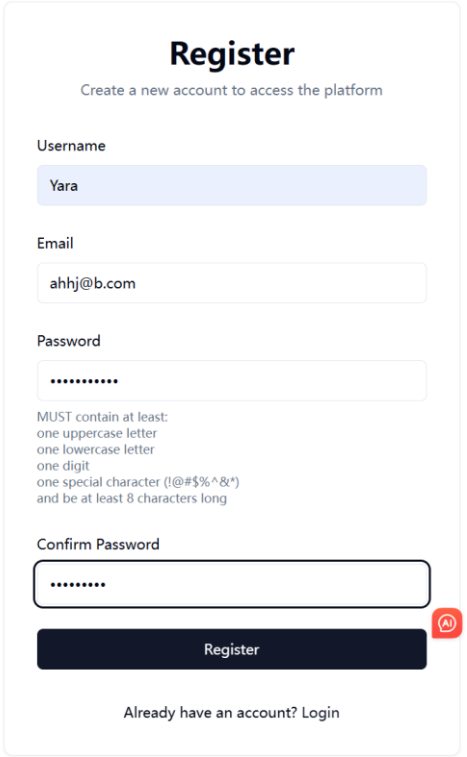
Defect ID	Test case	Title	Module	Severity	Steps	Expected Results	Actual Results	Suggestion
Defect_14	REG_3.6	Register and check email length boundaries	Register—Email Input	Medium	1. Enter an extremely long email that is prepared 2. If the system did not warn us, continue to add more characters 3. Click Register	The system should reject an email whose Length exceeds the limit	The system did not set any email length constraints, and the system page crashes.	Add strict email length validation on both frontend and backend to prevent oversized input and avoid crashes.
Defect_15	Static check (White Box Testing)	Error handler fails to handle network errors and 500 responses gracefully	Login – Error Handling	High	1. Inspect error handler logic. 2. Verify only **429** is explicitly checked. 3. Check behavior if backend returns **500** HTML or if	Robust error handler that handles 500 and network failures cleanly.	Toast may show an undefined message or fail when the response structure is not	Update the error handler to safely handle undefined responses and all status codes (including 500

					error.response is undefined.		expected.	and network failures) with a clear fallback message.

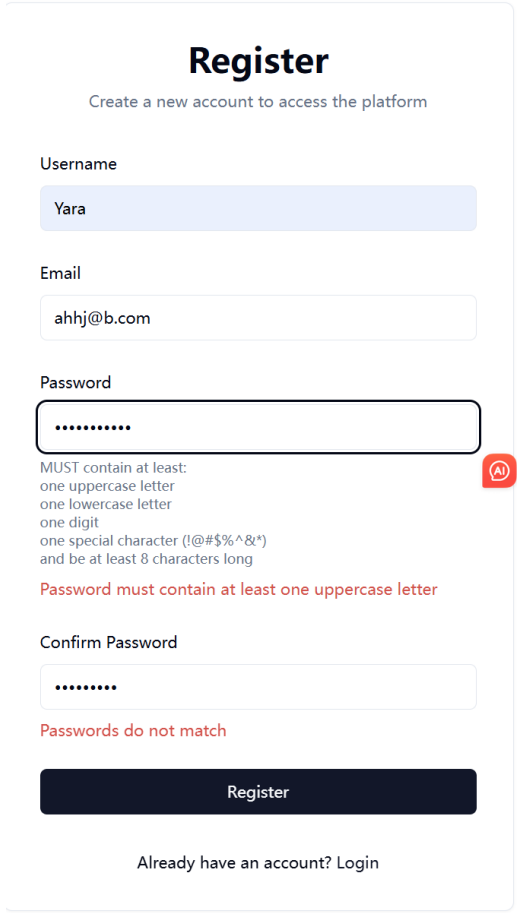
Attachment:

Defect_01:

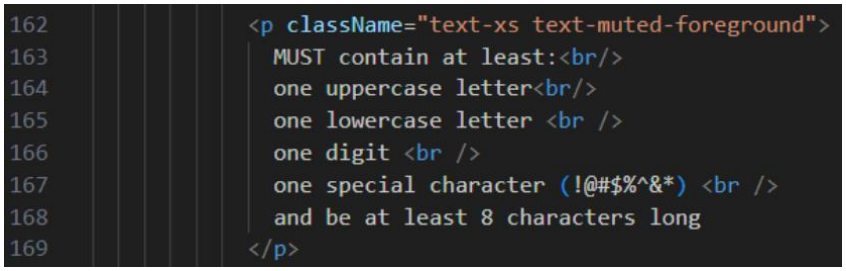
Before clicking **Register** and inputting the invalid password, there is no invalid hint.



After clicking **Register**--> The invalid hints appear:

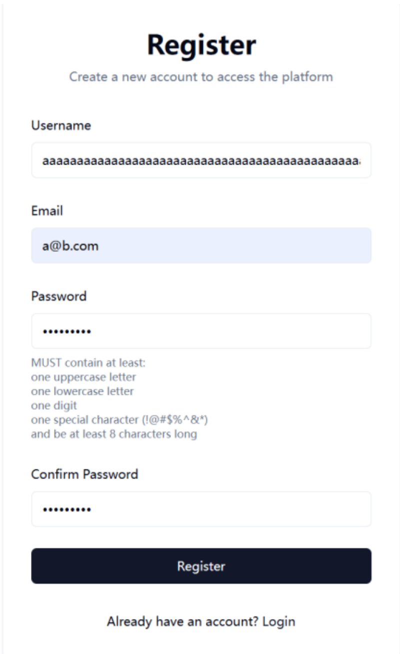


Static check:



Defect_02:

Before clicking **Register** and inputting **51** characters in the Username field:



After clicking **Register**--> Invalid hints appear:

Register

Create a new account to access the platform

Username

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Username must be at most 50 characters

Email

a@b.com

Password

•••••••••

MUST contain at least:
one uppercase letter
one lowercase letter
one digit
one special character (!@#$%^&*)
and be at least 8 characters long

Confirm Password

•••••••••

Register

Already have an account? Login

Defect_03:

Register with all invalid information, and click **Register**--> Only show Email is invalid:

Register

Create a new account to access the platform

Username

Enter your username

Email

1

请在电子邮件地址中包括“@”。“1”中缺少“@”。

Pas

•••

MUST contain at least:
one uppercase letter
one lowercase letter
one digit
one special character (!@#$%^&*)
and be at least 8 characters long

Confirm Password

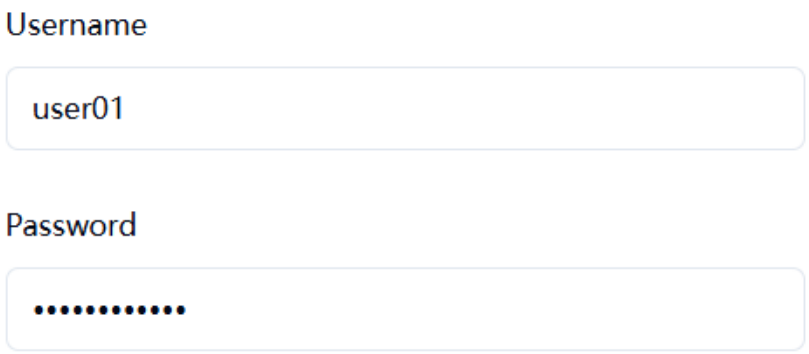
•••

Register

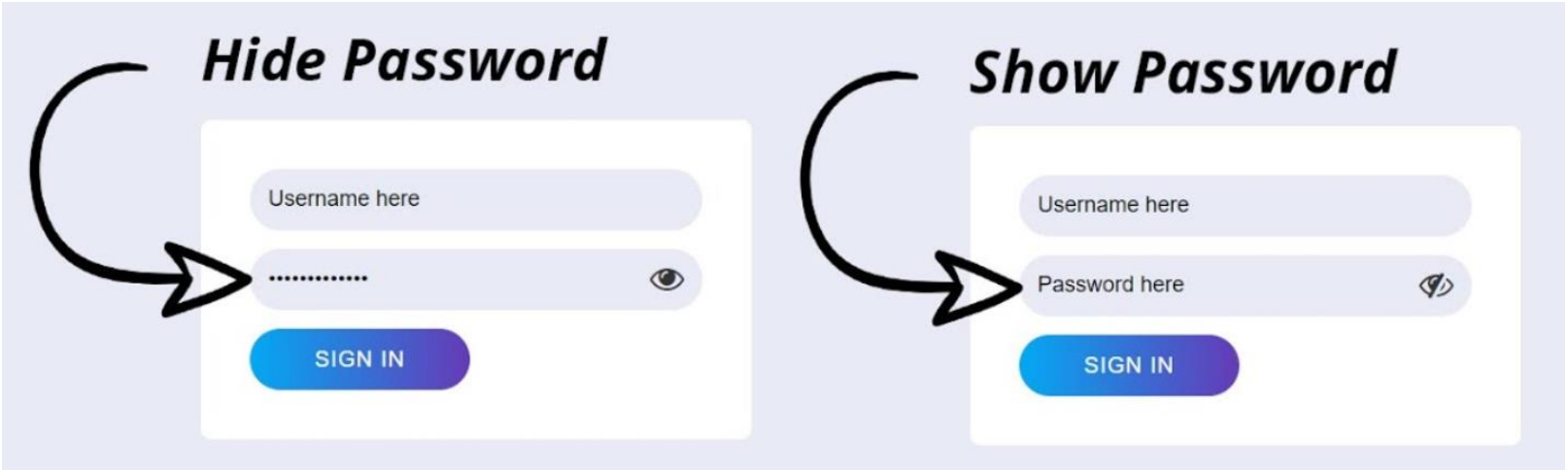
Already have an account? Login

Defect_04:

This website has no **mask** setting:



The other website with a password **mask**:



Static check:

```
161                 <Input
162                   id="password"
163                   type="password"
164                   placeholder="Enter your password"
165                   {...register('password')}
```

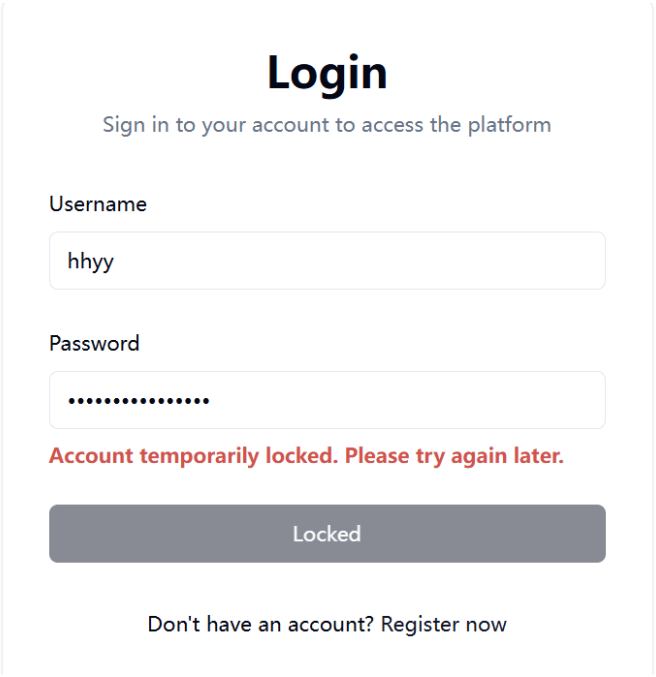
Defect_05:

Static check:

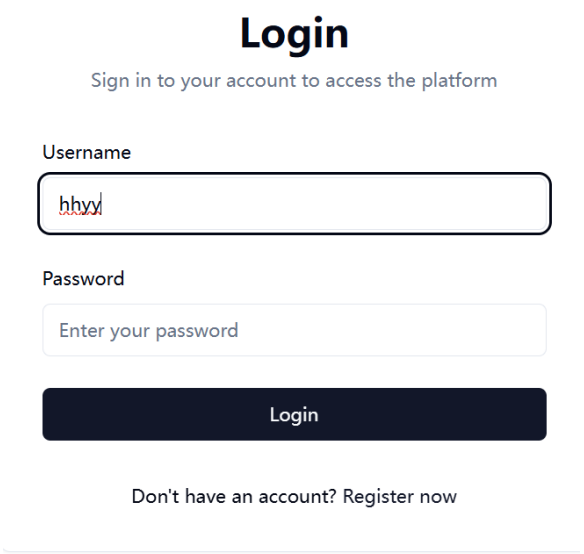
```
70    const {
71      register,
72      handleSubmit,
73      formState: { errors },
74    } = useForm<RegisterFormValues>({
75      resolver: zodResolver(registerSchema),
76    })
```

Defect_06:

Log in 5 times in one **IP-->Locked**

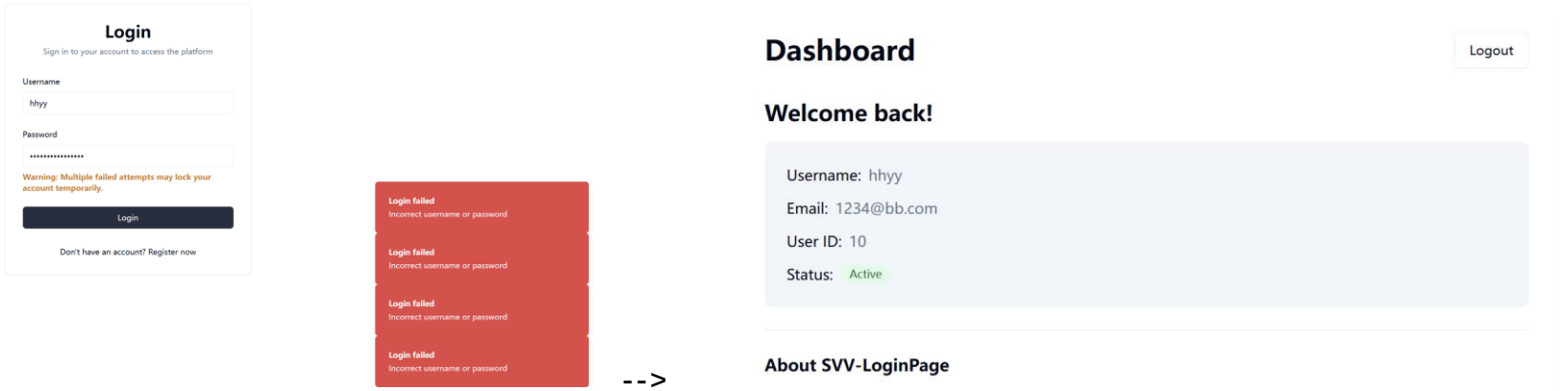


Change to another **IP**:

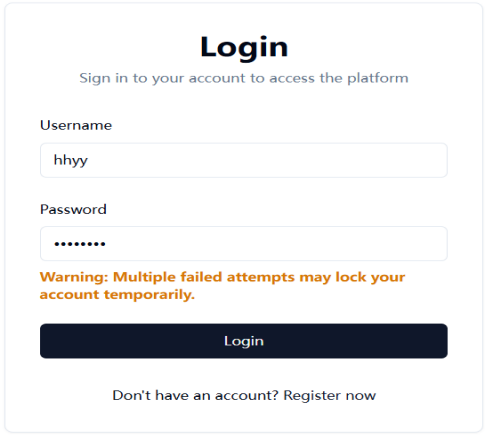


Defect_07:

Log in with wrong information **4** times **->** Log in with **correct information.**



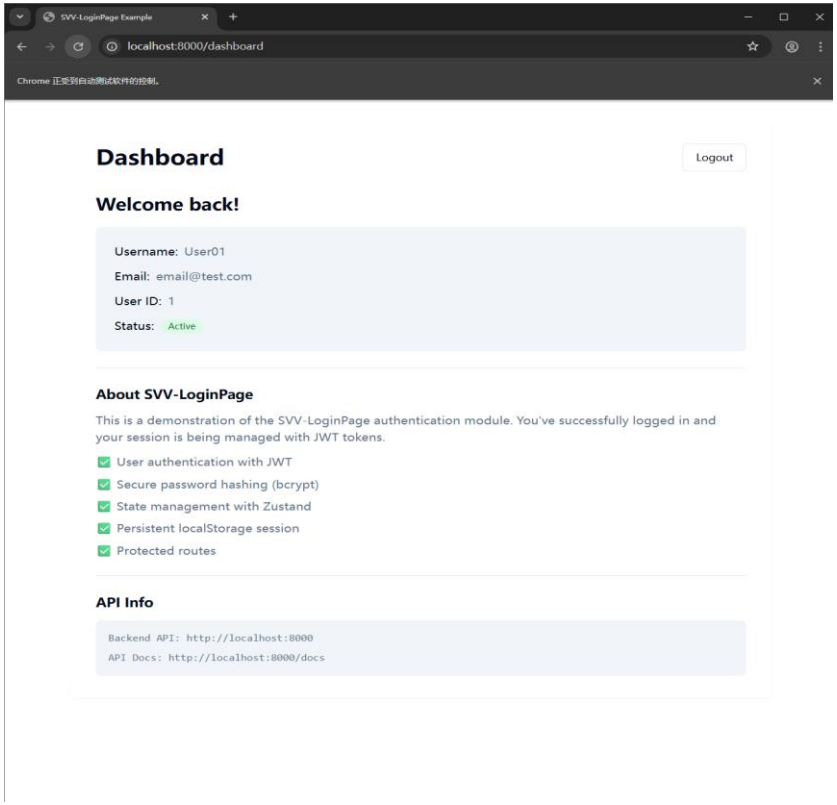
After **Logging out--> Try the 5th time to log in with wrong information,** the system **did not lock the account.**



Defect_08:

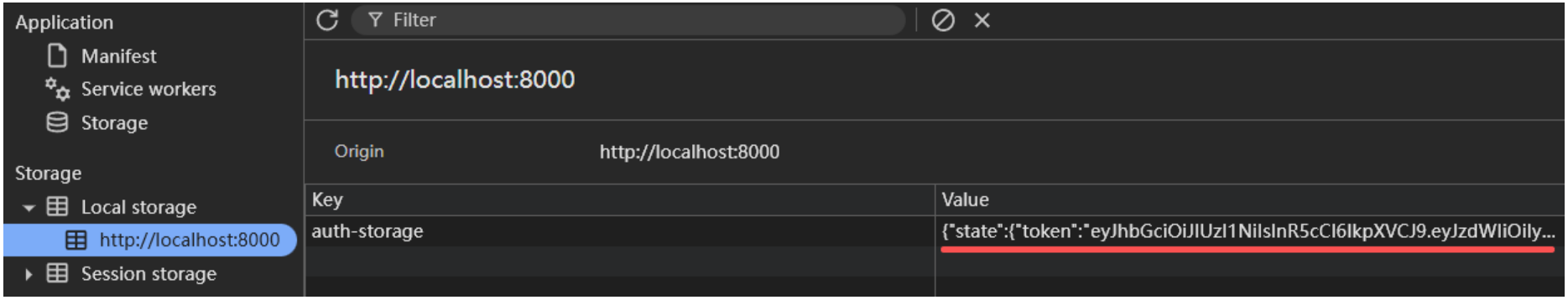
```
cursor.execute("DELETE FROM users WHERE username = 'User01';")
conn.commit()
```

After several times of page **refreshes,** the system did **not log the user out.**

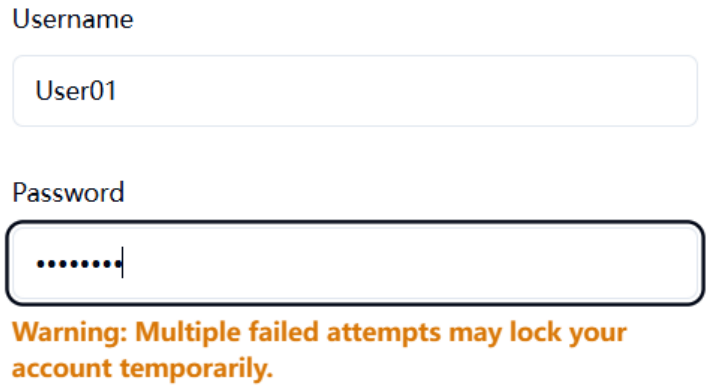


Defect_09:

Enable one successful login, press**'F12'** to enter Developer Tools and find Local Storage, copy the access token.



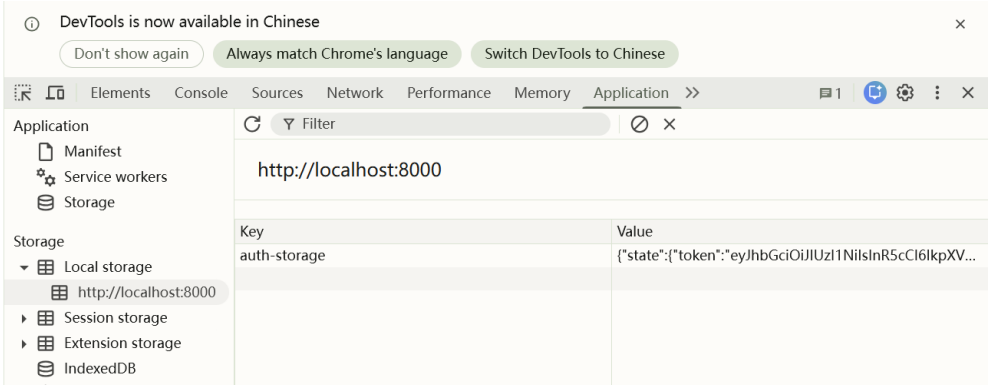
After deleting the user from the database, it **cannot login** successfully with old password:



However, if we used the in terminal and tried to login, the system still returns , it means the token is still valid, which it should not.

Defect_10:

The raw **JWT access token** is clearly visible in **Local Storage**. Token is accessible to any injected JavaScript, creating a severe **XSS** exploitation risk



Defect_11:

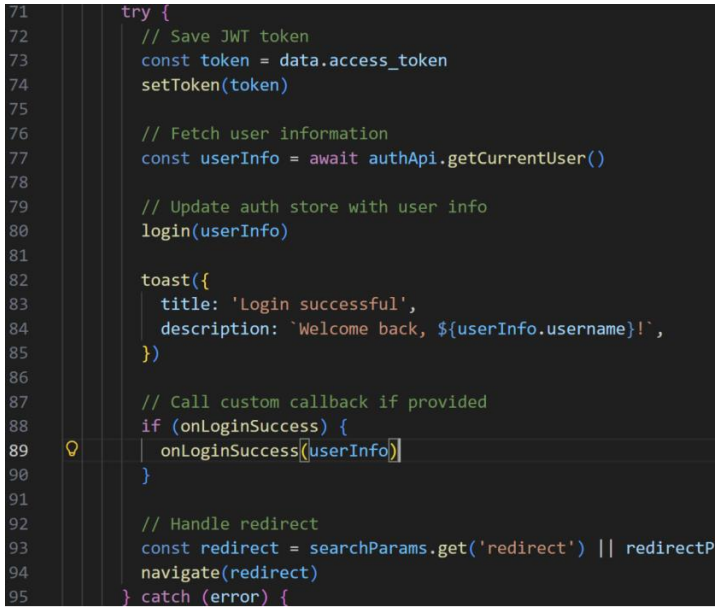
```
# JWT Configuration
secret_key: str = os.getenv(
    "SECRET_KEY",
    "09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"
)
algorithm: str = os.getenv("ALGORITHM", "HS256")
access_token_expire_minutes: int = int(os.getenv("ACCESS_TOKEN_EXPIRE_MINUTES", "30"))
```

Defect_12:

Static check:

```
95        } catch (error) {
96          console.error('Error fetching user info:', error)
97          toast({
98            variant: 'destructive',
99            title: 'Login failed',
100           description: 'Error retrieving user information',
101         })
102       } finally {
103         setIsLoading(false)
104       }
```

```
} catch (error) {
  console.error('Error fetching user info:', error)
  toast({
    variant: 'destructive',
    title: 'Login failed',
    // ...
  })
  // 缺陷位置：这里缺少了 clearToken() 或者 logout()
}
```



```
71      try {
72        // Save JWT token
73        const token = data.access_token
74        setToken(token)
75
76        // Fetch user information
77        const userInfo = await authApi.getCurrentUser()
78
79        // Update auth store with user info
80        login(userInfo)
81
82        toast({
83          title: 'Login successful',
84          description: `Welcome back, ${userInfo.username}!`,
85        })
86
87        // Call custom callback if provided
88        if (onLoginSuccess) {
89          onLoginSuccess(userInfo)
90        }
91
92        // Handle redirect
93        const redirect = searchParams.get('redirect') || redirectP
94        navigate(redirect)
95      } catch (error) {
```

Defect_13:

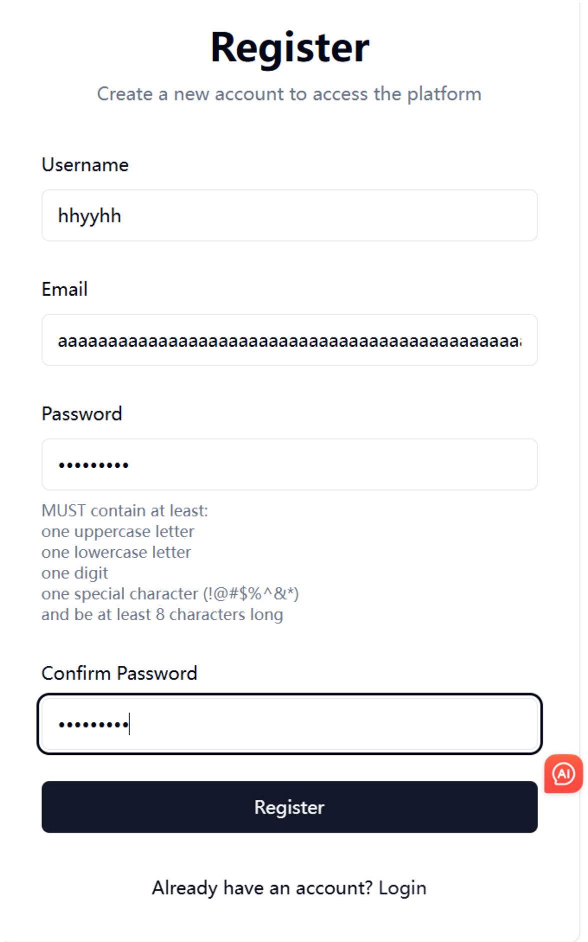
Static check:

isLockedOut does not persist across refresh and gives misleading security feedback.

```
54      const [isLockedOut, setIsLockedOut] = useState(false)
```

Defect_14:

Register with **254** characters in the **Email** field:



After clicking **Register, --->Website crashes down**

Defect_15:

Static check:

The error handler does not correctly handle **500** server errors or network failures, leading to **undefined messages or crashes.**

```
108        const status = error.response?.status
```