

Option #1: Your Own UML Diagram

Scott Miner

Colorado State University – Global Campus

Abstract

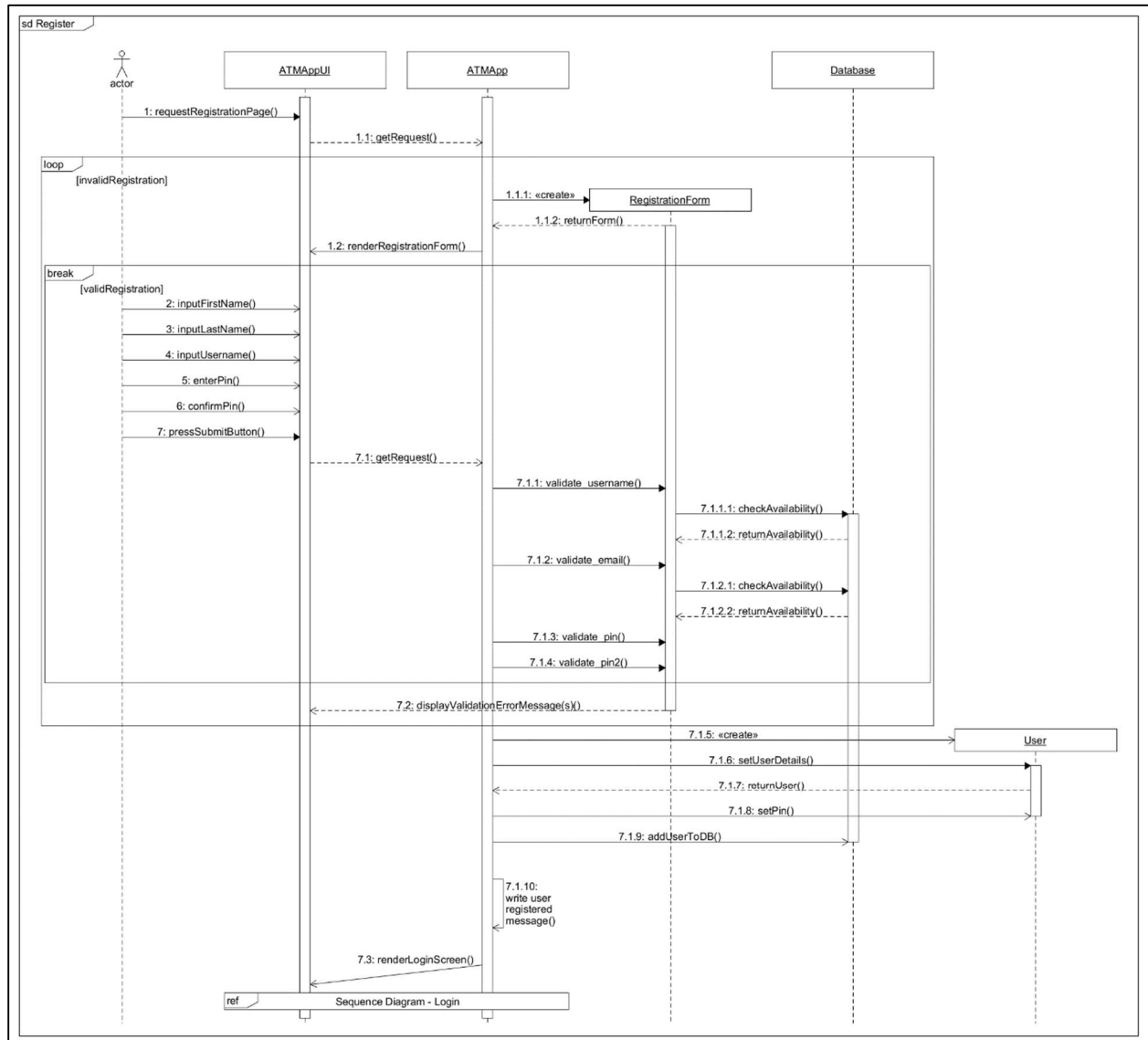


Figure 1. Sequence diagram depicting the registration use case

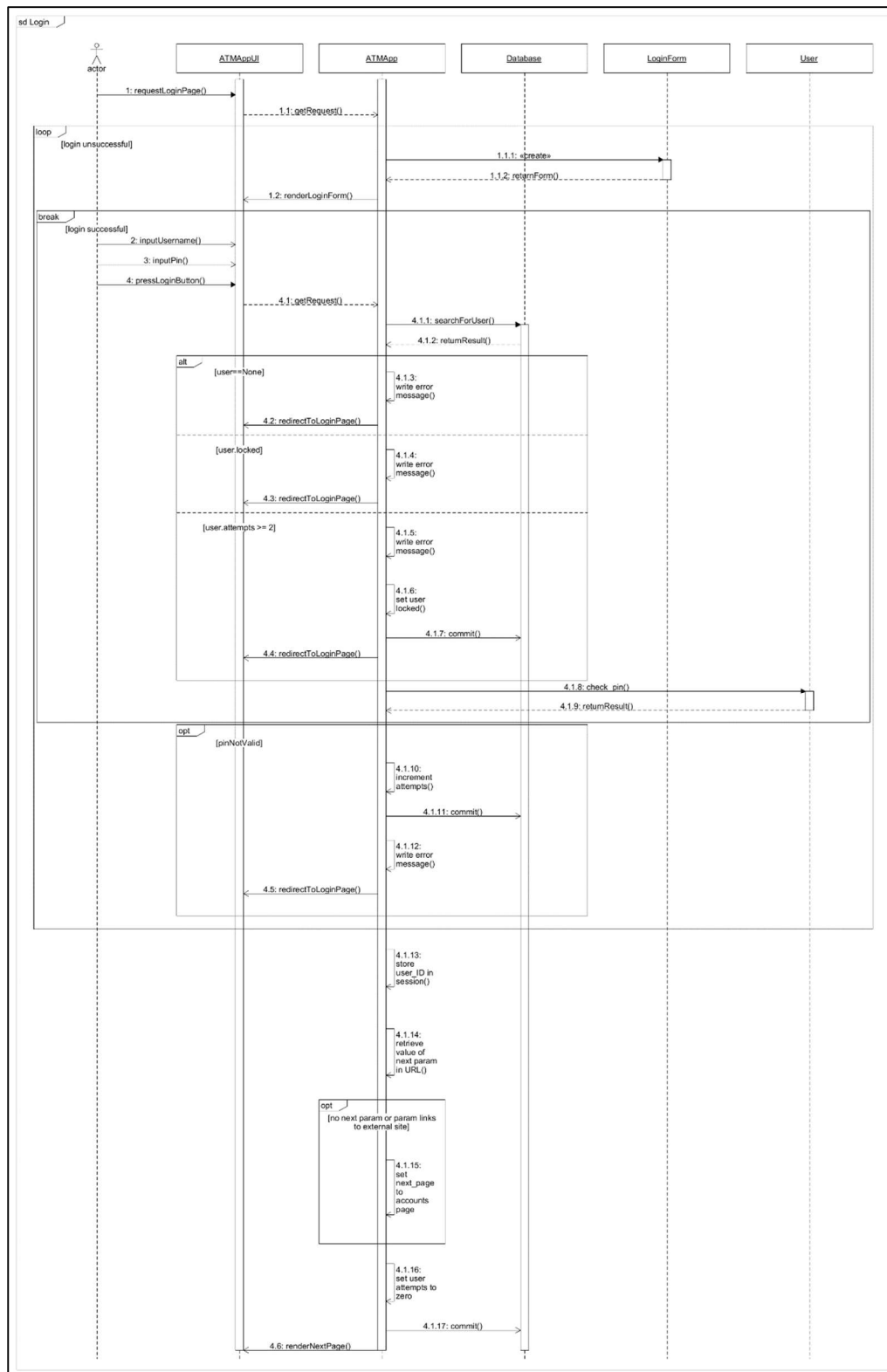


Figure 2. Sequence diagram depicting the login use case

```

<User ID: 1
First Name: Test
Last Name: User1
Username: TestUser1
Pin Hash:: pbkdf2:sha256:150000$EspAPFdT$9ce1a685bfe65452ccbf0d76fcf9f305406ca6c337790e291b7d2749c493f4b9
Email: test1@example.com
Attempts: 0
Locked: False>

<User ID: 2
First Name: Test
Last Name: User2
Username: TestUser2
Pin Hash:: pbkdf2:sha256:150000$C7VLn1aR$ff9e01014a246a9e8936bcec5258a2d78bd9ef495ef1ce6b3bd11ae116fae094
Email: test2@example.com
Attempts: 0
Locked: False>

<User ID: 3
First Name: Test
Last Name: User3
Username: TestUser3
Pin Hash:: pbkdf2:sha256:150000$uhChg0lF$008cd16baf0b42c9c73de5e0730b381d51c30b7c23bedf61ecbd7707d9c4d3d8
Email: test3@example.com
Attempts: 0
Locked: False>

<User ID: 4
First Name: Test
Last Name: User4
Username: TestUser4
Pin Hash:: pbkdf2:sha256:150000$hty08U4v$457ac4ebdc193030083a7210da85044a3777f559b7159177058375eb70c318c7
Email: test4@example.com
Attempts: 0
Locked: False>

<User ID: 5
First Name: Test
Last Name: User4
Username: TestUser5
Pin Hash:: pbkdf2:sha256:150000$bnL1mQ9k$ef5a9fe7f38c41a1f1922ee53f1b826f38d66289184036db34a6a1d40317f53a
Email: test5@example.com
Attempts: 0
Locked: False>

```


Figure 3. PIN hashes of pre-populated users in the database


Figure 4. Blank registration form


BankingApp Register Login


Register


Create your account. It's free and only takes a minute.












Pin can only contain digits.

×

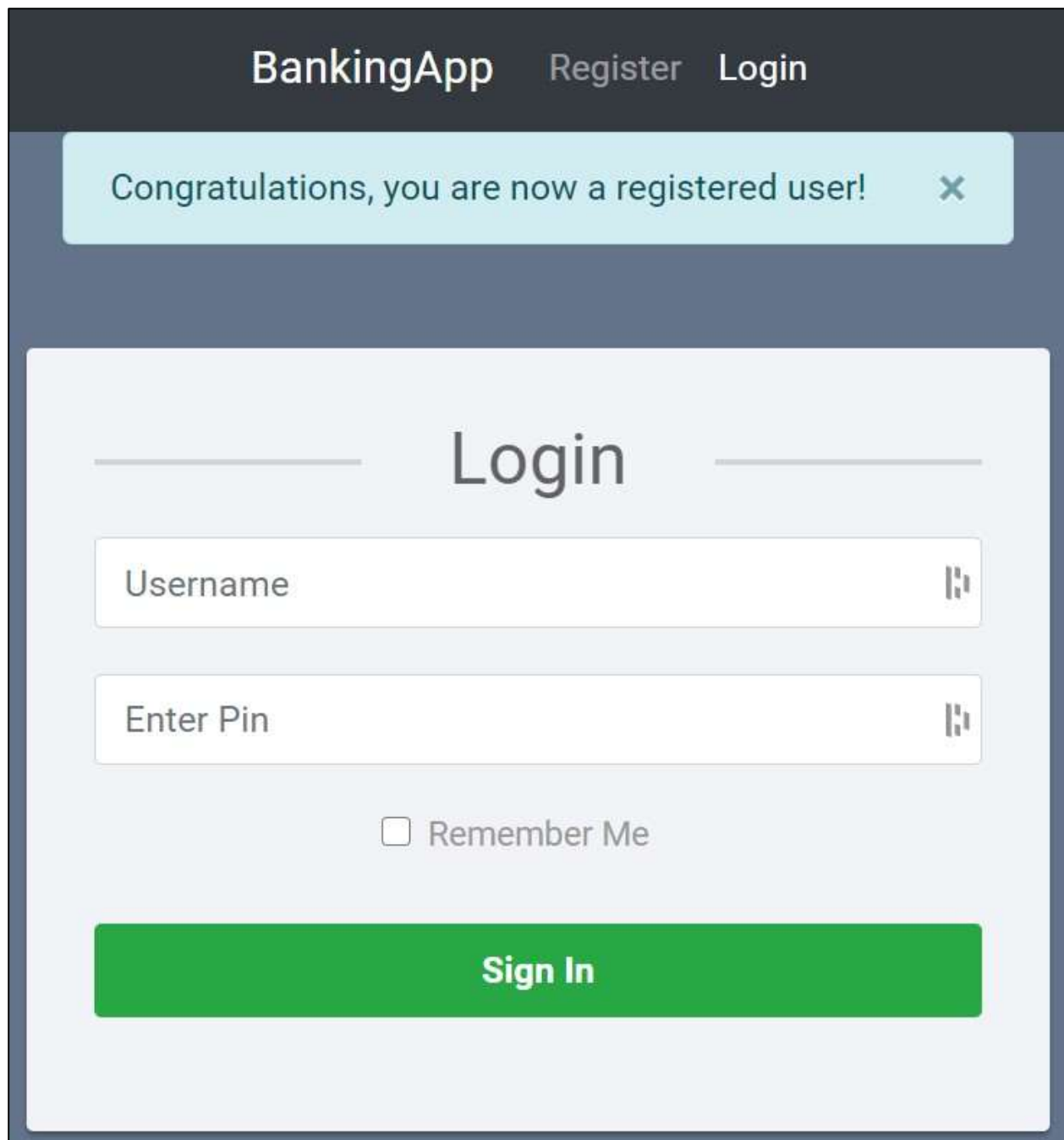


Field must be equal to pin.

×

Register Now

Figure 5. Registration form with returned errors



The image shows a mobile application interface for 'BankingApp'. At the top, there is a dark header bar with the app name 'BankingApp' in white, and two links, 'Register' and 'Login', in a lighter color. Below the header, a light blue notification banner displays the message 'Congratulations, you are now a registered user!' with a close button (an 'X' icon) on the right. The main content area is a light gray box with a white border. Inside this box, the word 'Login' is centered at the top, flanked by horizontal lines. Below 'Login', there are two input fields: 'Username' and 'Enter Pin'. Each field has a small icon on the right side, resembling a document or a key. Below the input fields, there is a checkbox labeled 'Remember Me'. At the bottom of the white box, there is a large green button with the text 'Sign In' in white.

Figure 6. Login screen after a successful registration

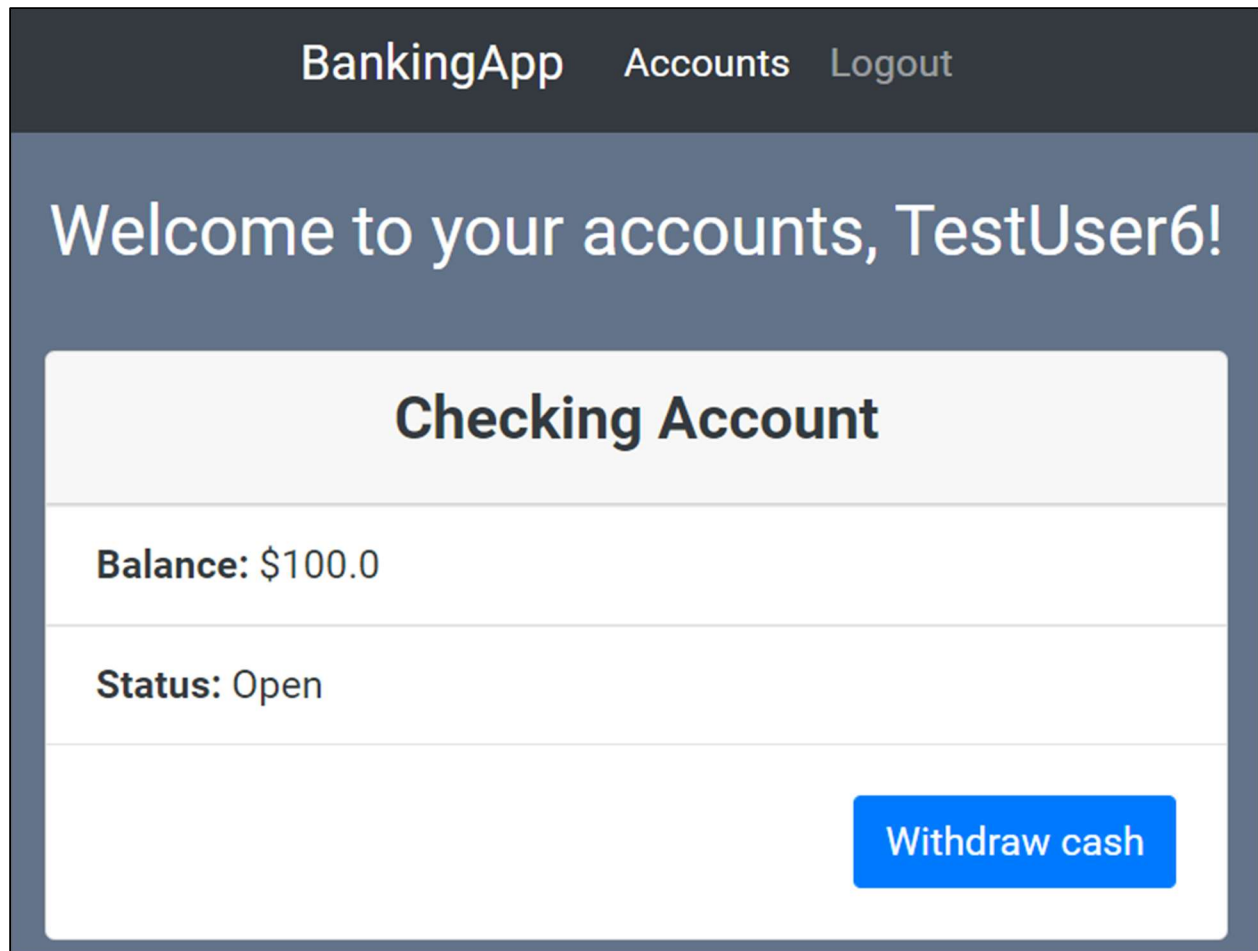


Figure 7. Checking account with a balance of \$100.00 and a status of "Open"

BankingApp [Accounts](#) [Logout](#)

How much would you like to withdraw, TestUser6?

Checking Account

\$20

\$40

\$60

\$80

Other

Submit

Figure 8. Withdrawal.html page

How much would you like to withdraw, TestUser6?

Not enough money in account

Checking Account

\$20

\$40

\$60

\$80

Other

Submit

Figure 9. Error received if the amount a user wishes to withdraw exceeds that which is in their account

BankingApp Accounts Logout

Welcome to your accounts, TestUser6!

Checking Account

Balance: \$0.0

Status: Closed

Figure 10. A closed checking account with an account balance of zero

The image shows a mobile application interface for a banking app. At the top, there is a dark header bar with the text "BankingApp" in white, followed by "Register" and "Login" in a lighter color. Below the header, a light blue error message box with a close button (X) displays the text "Invalid username or password". The main content area is a light gray box with the title "Login" centered at the top. Below the title, there are two input fields: "Username" and "Enter Pin", each with a small icon on the right side. Below the input fields, there is a checkbox labeled "Remember Me". At the bottom of the form, there is a large green button with the text "Sign In" in white.

Figure 11. Error message received after an erroneous login attempt

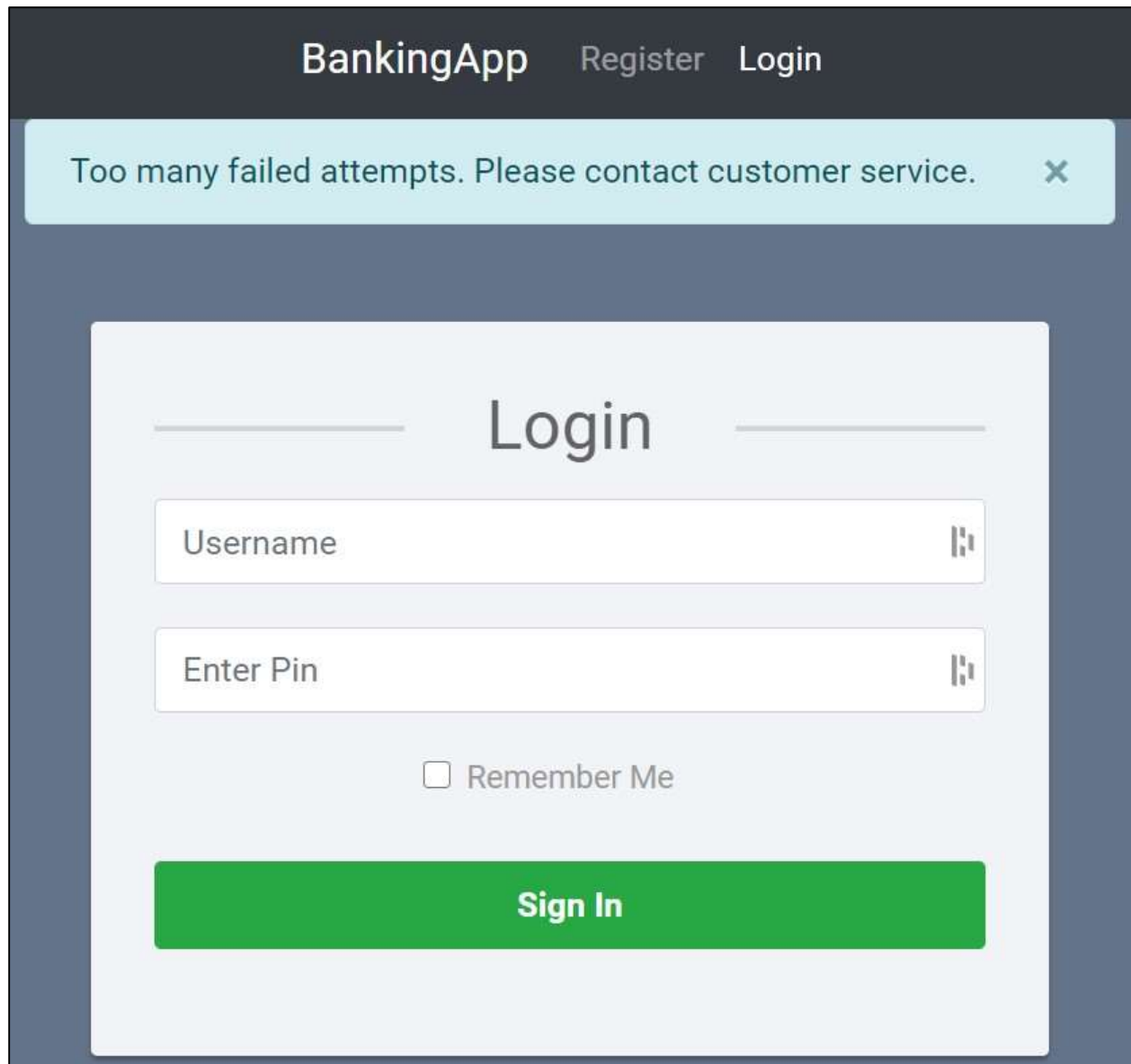


Figure 12. User account locked after three failed attempts

```
<User ID: 1
First Name: Test
Last Name: User1
Username: TestUser1
Pin Hash:: pbkdf2:sha256:150000$EspAPFdT$9ce1a685bfe65452ccbf0d76fcf9f305406ca6c337790e291b7d2749c493f4b9
Email: test1@example.com
Attempts: 3
Locked: True>

<User ID: 2
First Name: Test
Last Name: User2
Username: TestUser2
Pin Hash:: pbkdf2:sha256:150000$c7VLn1aR$ff9e01014a246a9e8936bcec5258a2d78bd9ef495ef1ce6b3bd11ae116fae094
Email: test2@example.com
Attempts: 0
Locked: False>
```

Figure 13. User with ID 1 is locked in the database after three failed attempts

Option #1: Your Own UML Diagram

For his Final Project in CSC505: Principles of Software Development, the student creates a sequence diagram for an automated teller machine (ATM). The following requirements comprise the software: (a) the customer must pass authentication before withdrawing money, (b) authentication is performed by checking a personal identification number (PIN), (c) the PIN can be correct or incorrect, (d) unsuccessful attempts are counted, (e) if the counter exceeds a limit, the customer is rejected, and (f) if the account balance is zero, then the account is closed.

The student implemented his application as a WebApp to gain more experience with WebApp design. In the real world, an ATM would require customers to input their debit cards before making withdraws, which means customers must have pre-existing accounts. To simulate this scenario in his WebApp, the student created a registration page, allowing users to create accounts from which they can withdraw money. Each user who registers with the service gets a free checking account with \$100.00 deposited. The student also pre-populated the application's database with five fictitious users via the *manage.py* file. One can load these fictitious users into the database by running the following command: *python manage.py seed*. Figure 1 shows a UML sequence diagram depicting the “registration” use case, whereas Figure 2 displays a sequence diagram depicting the “login” scenario.

Pressman and Maxim (2020) define *secure coding* as coding in such a way as to not introduce vulnerabilities into the code. Likewise, Win *et al.* (2002) write that secure coding involves developers taking on a defensive attitude, thinking both about how to implement software functionality and how to do so in such a way as to not allow attackers to take advantage of developers' source code for negative purposes. Moreover, according to Gasiba *et al.* (2021),

the “financial impact of attacks on Cyberphysical systems will exceed \$50 billion in 2023” (p.

1). Therefore, when developing software for an ATM, security is of the utmost importance.

The student implemented security safeguards in his WebApp in three specific ways: (a) by disabling attackers from redirecting URLs to external websites, (b) by hashing PINs, and (c) by locking out customers who have attempted to login three or more times unsuccessfully. The first scenario is the most difficult to explain. The *@login_required* decorator from the *Flask-Login* extension will redirect unauthenticated users to the login page when they attempt to access views protected with this decorator, such as the *accounts.html* page. The decorator adds a query string argument to the URL so that if an unauthorized user attempts to access the accounts page, for instance, the redirected URL would appear as */login?next=/accounts*, indicating the application should load this page for the user upon login. However, attackers could use a similar technique to redirect users to external, malicious websites. To prevent such attacks, the application validates the “next” argument’s contents, ensuring users can only be redirected to relative URL paths and none that are absolute, as shown in step 4.1.15 of Figure 2 (Grinberg, 2018).

Chang *et al.* (2014) describe password hashing as a technique used to protect clients’ passwords by transforming them into altogether different strings called hashes. Figure 3 shows the hashed PINs of all pre-populated users in the database. By storing the PINs as hashes rather than literal values, the application disables any would-be attackers from obtaining users’ credentials should they unexpectedly gain access to the database. Lastly, as shown in Figure 3, users also have the fields “attempts” and “locked” associated with their accounts. If a user enters the wrong credentials three or more times, their account becomes locked, and they will need to contact their bank to remove the lock. On the other hand, if a user enters his credentials correctly

on any attempt before the third, his attempts are reset to zero, as shown in Step 4.1.16 of Figure 2. These examples demonstrate a few of the ways the student implemented security safeguards in his ATM application.

The basic workflow of the application proceeds as follows. Upon navigating to the home screen, the user is directed to the application's registration page, where users are asked to input their first name, last name, username, email address, and PIN. PINs can be anywhere between 4 and 8 digits long and may only contain numbers. An additional field on the registration page asks users to confirm their PINs to ensure they are entered correctly. Figure 4 shows a blank registration form, while Figure 5 shows a few of the errors a user may receive during an unsuccessful registration attempt. After registering successfully, the user is redirected to the login page, as shown in Figure 6. Also, Figure 6 shows the confirmation message a user receives upon successful registration: "Congratulations, you are now a registered user."

Figure 7 shows the *accounts.html* page, displaying what a new user sees upon successfully logging in for the first time: a checking account with \$100.00 deposited. From this image, one can see the checking account's status is set to "Open," and the user has the option to withdraw money from the account. Upon clicking the "Withdraw Cash" button, the application renders the *withdrawal.html* page, where users can withdraw cash in amounts of \$20, \$40, \$60, and \$80. Also, users have the option to specify an "other" amount less than \$300.00. If the amount the user enters exceeds that which is in their bank account, the user receives the error message shown in Figure 9. If a user withdraws all the money from their account, the account's status is set to "Closed," as shown in Figure 10. These sequences of events depict the ATM application's main functionality.

Additionally, Figure 11 shows what happens if a user enters their credentials incorrectly: they receive an error message stating, “Invalid username or password.” Step 4.1.10 of Figure 2 indicates that the application increments a counter for each unsuccessful login attempt. If a user enters their credentials three or more times unsuccessfully, the application locks their account, and they receive the error message shown in Figure 12: “Too many failed attempts. Please contact customer service.” Figure 13 shows a user’s account locked in the database after three failed attempts.

In conclusion, this paper provides an overview of the student’s ATM application. The paper presents two UML sequence diagrams, one for the “registration” use case and another for that of the “login,” as well as a series of screenshots depicting the successful execution of the program. The ATM WebApp requires a customer to pass authentication before withdrawing money, performs authentication by checking a PIN, determines whether the PIN is correct, counts unsuccessful login attempts, and locks a user’s account after three unsuccessful attempts. New customers are awarded \$100.00 for registering with the application, and customers have the option to withdraw money in the amounts of \$20, \$40, \$60, and \$80. If a customer withdraws all the money from their account, their account is marked as “Closed.” Finally, the WebApp implements three security safeguards to help protect users’ sensitive information: (a) disabling redirects to absolute URLs, (b) hashing user PINs, and (c) locking the accounts of users who have attempted three or more unsuccessful logins. These elements demonstrate the WebApp’s main functionality.

References

- Chang, D., Jati, A., Mishra, S., & Sanadhya, S. K. (2014). Rig: A simple, secure, and flexible design for password hashing. *International Conference on Information Security and Cryptology*, 361–381.
- Gasiba, T. E., Lechner, U., Pinto-Albuquerque, M., & Mendez, D. (2021). Is Secure Coding Education in the Industry Needed? An Investigation Through a Large-Scale Survey. *ArXiv:2102.05343 [Cs]*. <http://arxiv.org/abs/2102.05343>
- Grinberg, M. (2018, January 2). *The Flask Mega-Tutorial Part V: User Logins*.
<http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-v-user-logins>
- Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (Ninth edition). McGraw-Hill Education.
- Win, B. D., Piessens, F., Joosen, W., De, B., Frank, W., Joosen, P. W., & Verhanneman, T. (2002). *On the Importance of the Separation-of-Concerns Principle in Secure Software Engineering*.