Hacking the Vuln Bank Login (SQL Injection PoC)

Prepared By: LINTO BABY

1. Executive Summary

Quick Summary: This report covers my successful little project exploiting the intentionally vulnerable Vuln Bank app. The goal was to prove how dangerous **SQL Injection (SQLi)** is. I found a critical flaw in the login mechanism that let me completely bypass the admin password check on my first try! After sneaking into the administrator dashboard, I approved a test user's loan request to validate my access. This whole exercise was a perfect example of why developers **must use Prepared Statements** to keep user input from being executed as code.

2. Introduction

What's Going On: This is just a quick write-up of the work I did on the **Vulnerability Assessment of Vuln Bank**. I got the Docker environment set up and the app running successfully. The main objective was to **identify and exploit a common web application vulnerability** in a controlled lab environment. This really helped me get a **deeper practical understanding of defensive programming**—and it was a blast to execute!

3. Key Findings and Analysis

3.1. Project Milestones

Milestone	**Status**	**Notes**
Initial Setup & Configuration	Complete	Vuln Bank installed via Docker and test user account created.
Authentication Bypass Proof of Concept	**Complete**	**SQL Injection successfully exploited to gain administrative privileges.**
Access Validation (Loan Approval)	Complete	Loan request approved and bank balance confirmed on the test user account.

3.2. The Exploit: How I Got Admin Access

The Vulnerability (SQL Injection) Explained Simply:
The app was written in a way that mixes user input (the password I typed) directly into the database's instructions (the SQL query). This is a huge security mistake because it means I can change the instructions!
I used the following payload in the password field of the admin login: ' OR 1=1--

- **What it does:** This trick changes the database's question from, "Is the username correct **AND** the password correct?" to "Is the username correct **OR** is '1=1' true?"
- **The magic:** Since '1=1' is *always* true, the entire password check is bypassed.
- **The Cleanup:** The -- at the end is a comment marker that tells the database to ignore the rest of the original code, preventing any errors and letting me log right in.

Result:
I immediately gained full, unauthorized access to the administrator panel. This is a classic Broken Access Control issue. I then used this privileged access to approve my own pending loan request and confirm the updated balance on my user account. Happy days!

3.3. Lessons Learned (Risks and Mitigation)

- **Risk:** This vulnerability is simple to execute and incredibly dangerous. In the real world, this could allow an attacker to steal user data, modify account balances, or take down the entire system.
- **Mitigation Strategy (The Fix!):** The primary defense is implementing **Parameterized Queries (or Prepared Statements)**. This is the gold standard for secure coding. It forces the database to always treat user input as data, never as executable code, instantly killing the SQLi attack.

4. Conclusion and Recommendations

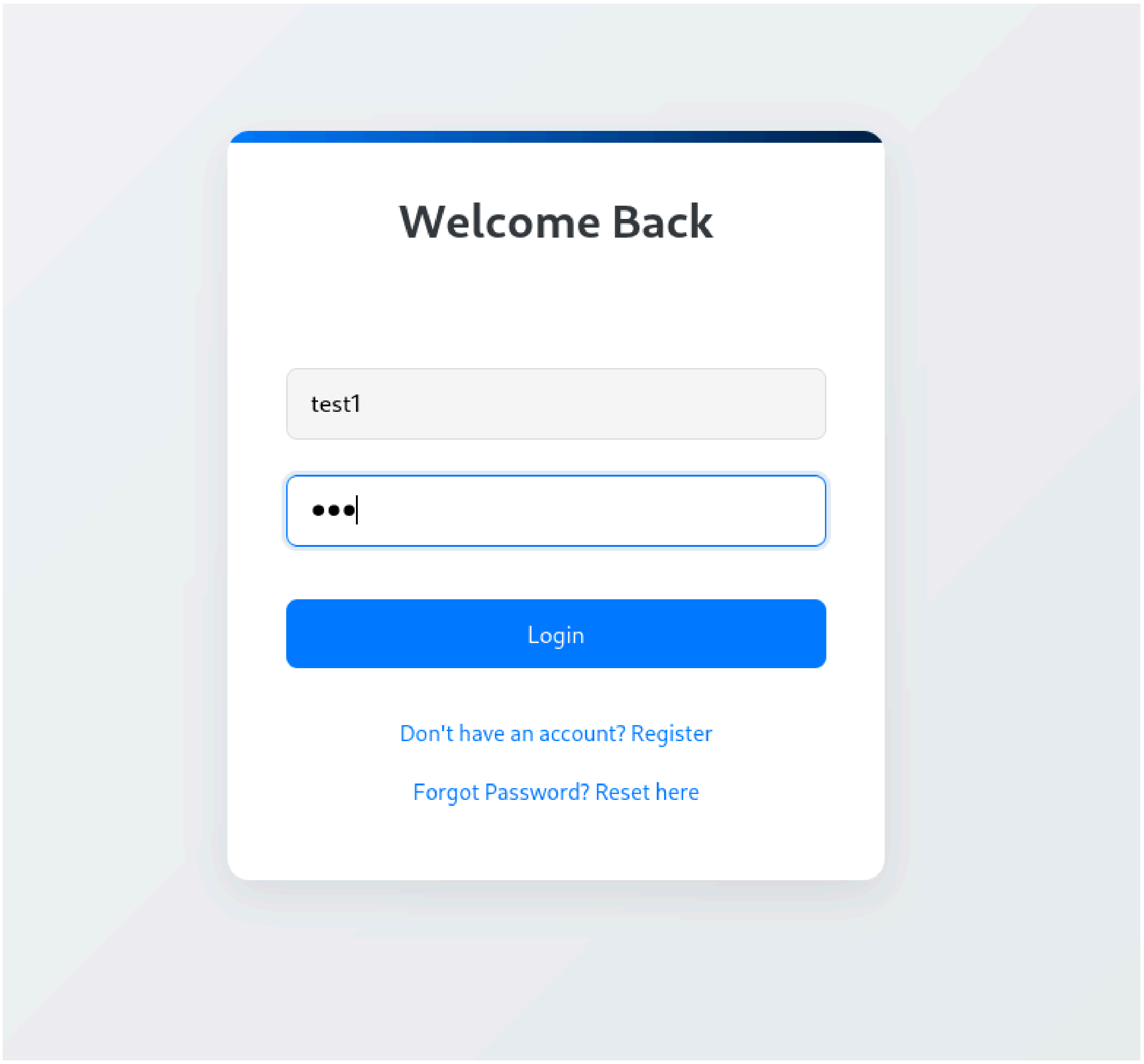
Conclusion: This test was a huge success! Exploiting the **SQL Injection** vulnerability was straightforward and demonstrated how easily a critical security flaw can exist in basic functionality like a login form. It was a great lesson in why defensive programming is essential.

Recommendations:

1. **Immediate Fix:** All database interactions **must** be updated to use **Parameterized Queries** to prevent this class of attack.
2. **Defense in Depth:** Implement strong **input validation** on all fields to filter out suspicious characters (like quotes and dashes) as an extra security layer.
3. **Next Steps:** Time to find the next weakness in the Vuln Bank application!
4. "I'm excited to keep learning so I can find and exploit the next vulnerability!", in future i will do it

5. Visual Appendices (Screenshots & Supporting

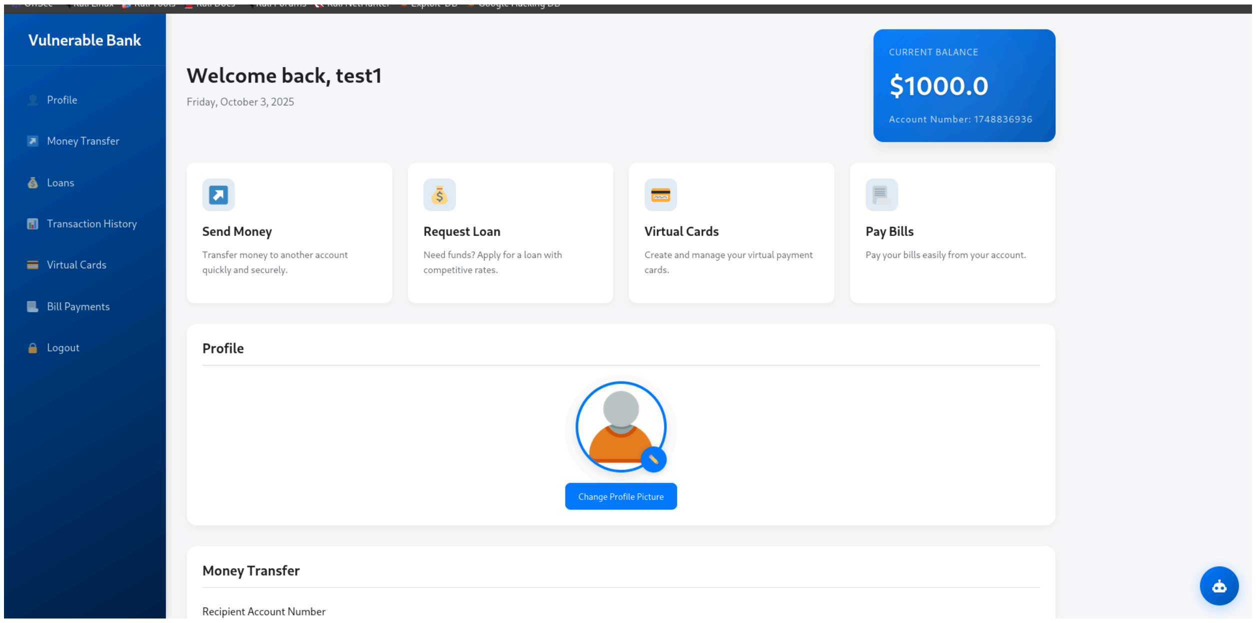
Images)



Welcome Back

test1

●●●



Vulnerable Bank

Profile
Money Transfer
Loans
Transaction History
Virtual Cards
Bill Payments
Logout

Welcome back, test1

Friday, October 3, 2025

CURRENT BALANCE

$1000.0

Account Number: 1748836936

Send Money

Transfer money to another account quickly and securely.

Request Loan

Need funds? Apply for a loan with competitive rates.

Virtual Cards

Create and manage your virtual payment cards.

Pay Bills

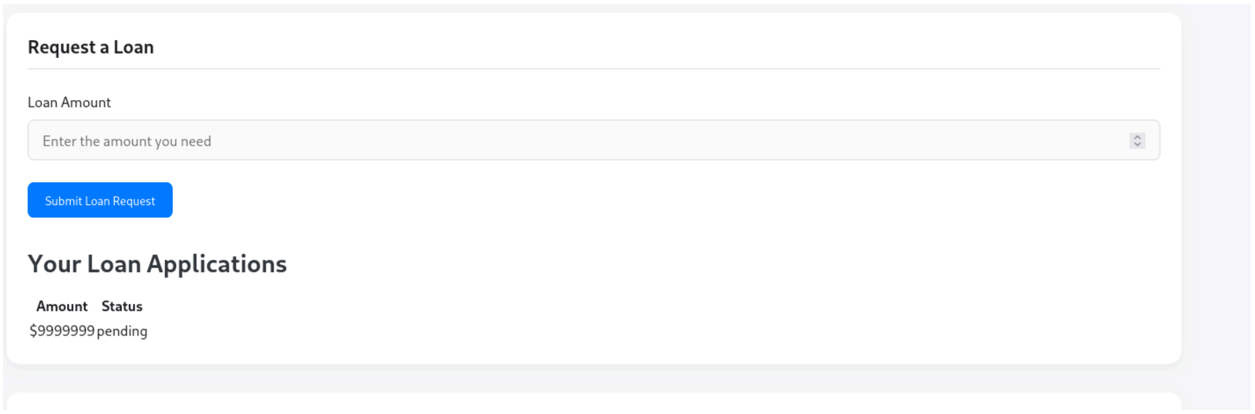
Pay your bills easily from your account.

Profile

Change Profile Picture

Money Transfer

Recipient Account Number



Request a Loan

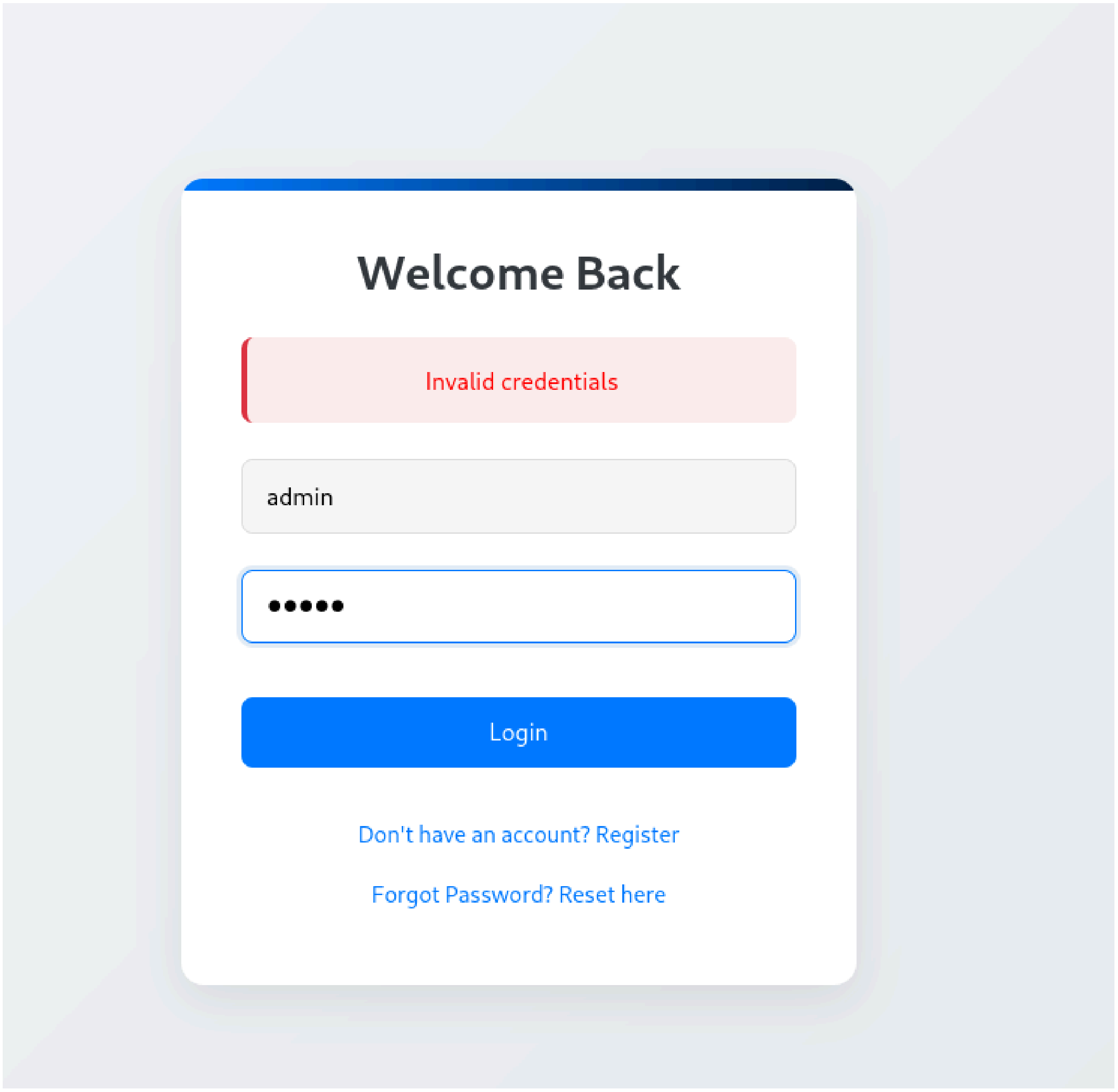
Loan Amount

Enter the amount you need

Submit Loan Request

Your Loan Applications

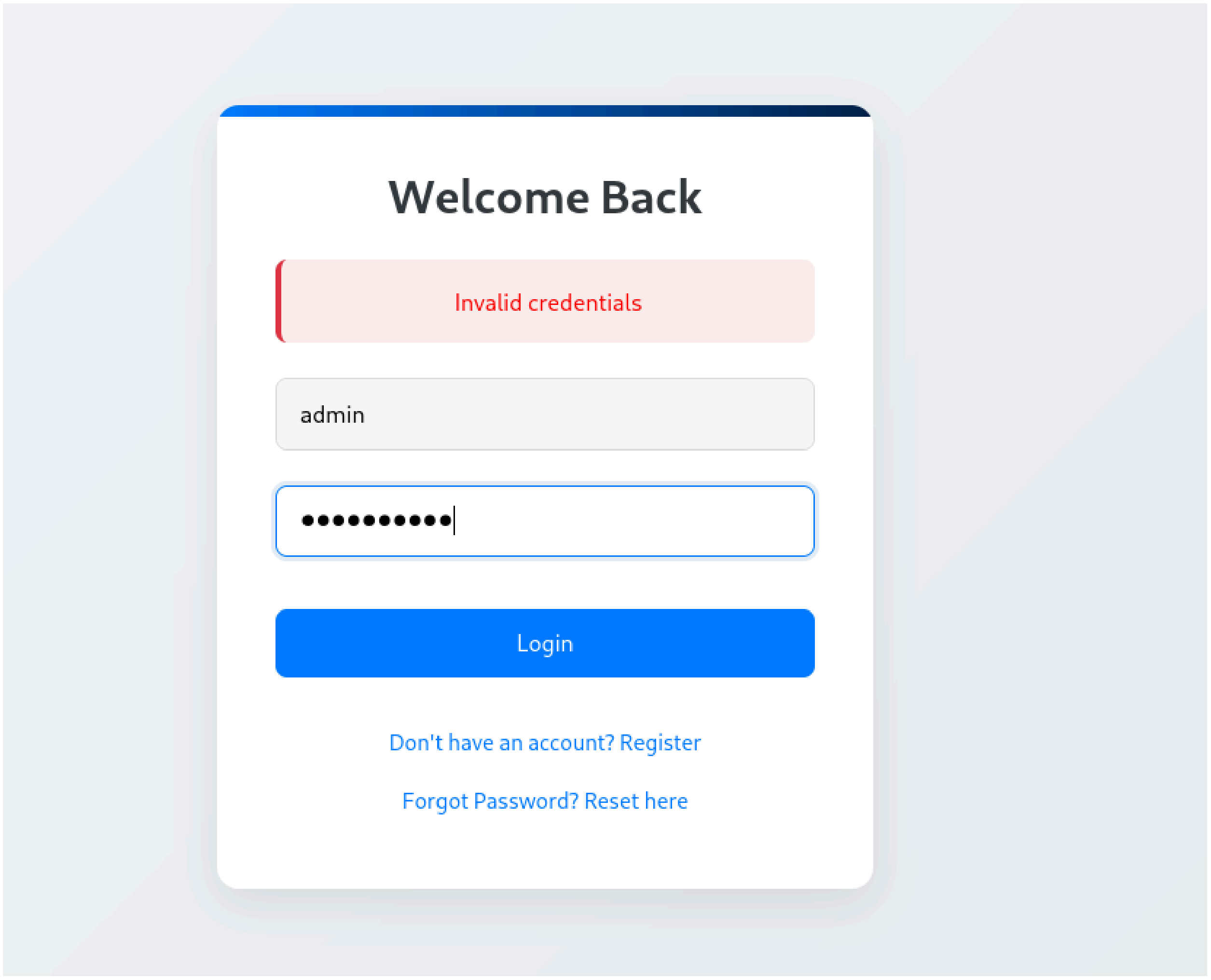
Amount	Status
$9999999	pending


Welcome Back

Invalid credentials

admin

●●●●●



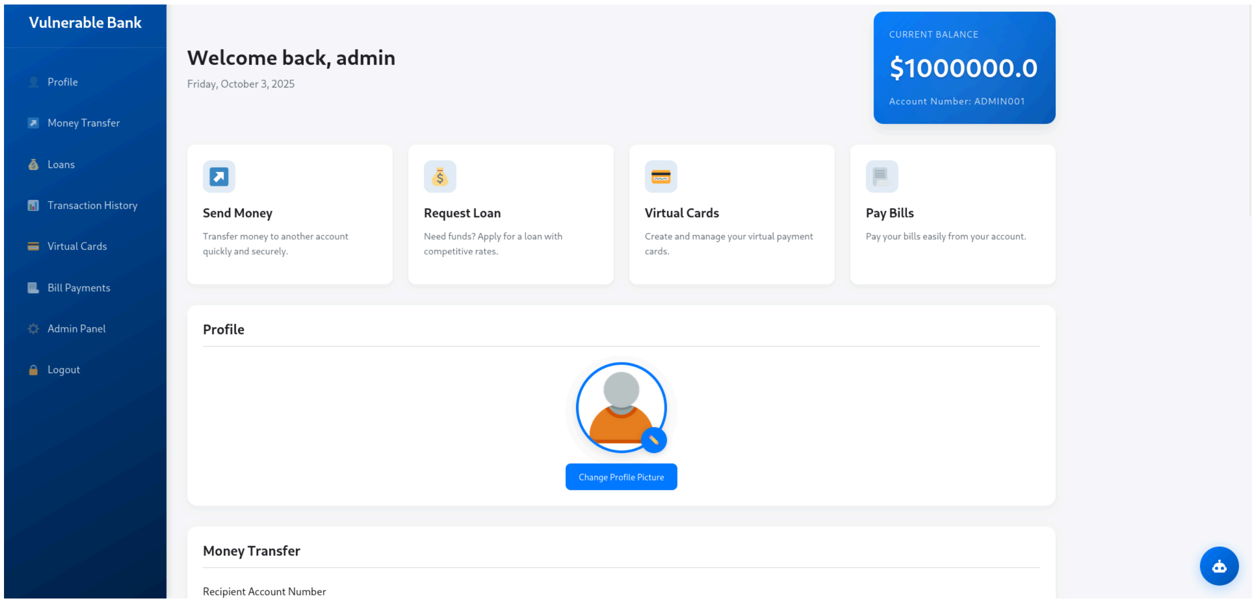
Welcome Back

Invalid credentials

admin

●●●●●●●●●●●

Query was ‘OR 1=1–



Welcome back, admin

Friday, October 3, 2025

CURRENT BALANCE

$1000000.0

Account Number: ADMIN001

Send Money
Transfer money to another account quickly and securely.

Request Loan
Need funds? Apply for a loan with competitive rates.

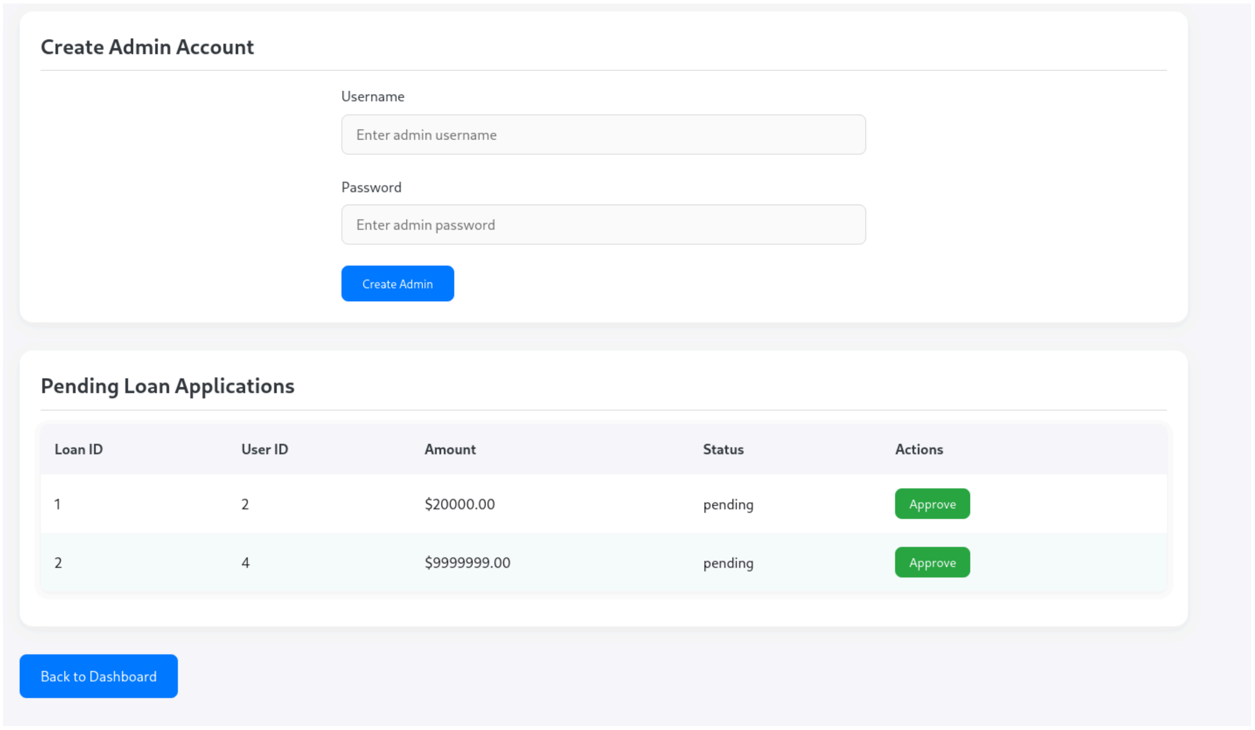
Virtual Cards
Create and manage your virtual payment cards.

Pay Bills
Pay your bills easily from your account.

Profile

Money Transfer

Recipient Account Number



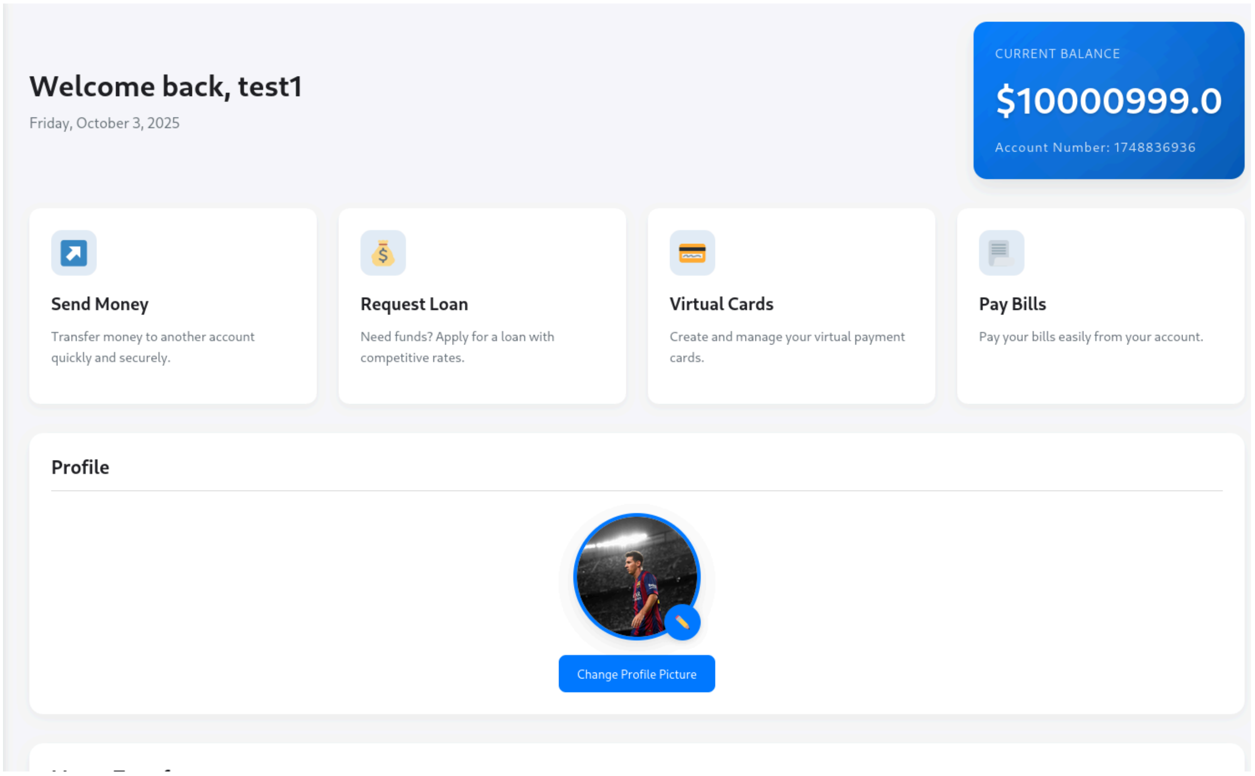
Create Admin Account

Username: Enter admin username

Password: Enter admin password

Pending Loan Applications

Loan ID	User ID	Amount	Status	Actions
1	2	$20000.00	pending	Approve
2	4	$9999999.00	pending	Approve



CURRENT BALANCE

$10000999.0

Account Number: 1748836936

Welcome back, test1

Friday, October 3, 2025

Send Money

Transfer money to another account quickly and securely.

Request Loan

Need funds? Apply for a loan with competitive rates.

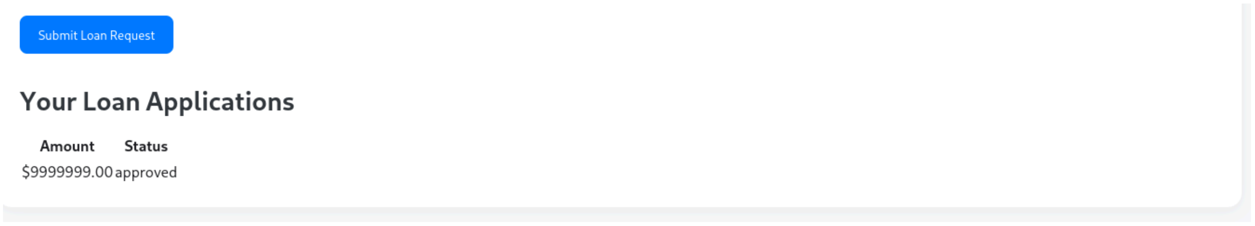
Virtual Cards

Create and manage your virtual payment cards.

Pay Bills

Pay your bills easily from your account.

Profile



Your Loan Applications

Amount	Status
$9999999.00	approved