**Title: ATM Simulation**

**Name (s):** Nandini Kapil , Charu , Aashutosh Srivastava , Anuj Mondal

**Reg No(s):** 24MCA0180,24MCA0192,24MCA0197,24MCA0216

**Faculty Name:** **Dr. KUMARESAN P**

## Problem Statement:

The Banking ATM Simulator was developed to create a realistic and user-friendly virtual ATM system that simulates the operations of a real-world ATM machine. Traditional banking requires physical presence at ATMs or bank branches for basic operations. This project addresses this limitation by providing a software solution that accurately replicates ATM functionality, allowing users to simulate banking operations in a secure environment. The simulator serves both educational purposes for understanding banking systems and practical applications for testing banking interfaces.

**Test Environment:** macOS 24.4.0 (Darwin)

**Python Version:** Python 3.13

## Libraries:

- Built-in Python Libraries:
- tkinter: For creating the graphical user interface
- json: For handling user data storage
- datetime: For timestamping transactions
- os: For file and directory operations
- functools: For partial function application in event handling

**Database:** The application uses a JSON file-based data storage system (users.json) to persist user account information, balances, and transaction history. This flat-file approach was chosen for simplicity and portability.

## Code:

The application consists of two main Python files:

1. **main.py (GUI Interface):**

```
import tkinter as tk
from tkinter import messagebox, ttk
from atm import ATM
import json
from functools import partial

# Modern UI constants
COLORS = {
    'bg_dark': '#212529',         # Dark background
    'bg_medium': '#343a40',       # Medium background
    'accent': '#6c757d',          # Accent color
    'text_light': '#f8f9fa',      # Light text color
    'success': '#228B22',         # Dark green for success
    'primary': '#0062cc',         # Dark blue for primary actions
    'danger': '#dc3545',          # Red for dangerous actions
    'warning': '#ffc107',         # Yellow for warnings
    'info': '#17a2b8',            # Cyan for info
    'muted': '#6c757d'            # Muted gray
}

# Create a custom style for widgets
def create_button(parent, text, command, bg_color=COLORS['primary'],
fg_color=COLORS['text_light'], width=15, height=2):
    """Create a modern styled button"""
    button = tk.Button(
        parent,
        text=text,
        command=command,
        font=('Helvetica', 12),
        width=width,
        height=height,
        bg=bg_color,
        fg=fg_color,
        activebackground=bg_color,
        activeforeground=fg_color,
        relief=tk.FLAT,
        borderwidth=0,
        highlightthickness=0,
        padx=10,
        pady=5
    )
    # Hover effect
```

```
43.     button.bind("<Enter>", lambda e:
    e.widget.config(bg=_adjust_lightness(bg_color, 1.1)))
44.     button.bind("<Leave>", lambda e: e.widget.config(bg=bg_color))
45.     return button
46. 
47. def create_entry(parent, show=None, width=20):
48.     """Create a modern styled entry field"""
49.     entry = tk.Entry(
50.         parent,
51.         font=('Helvetica', 12),
52.         width=width,
53.         bg=COLORS['bg_medium'],
54.         fg=COLORS['text_light'],
55.         insertbackground=COLORS['text_light'],  # Cursor color
56.         relief=tk.FLAT,
57.         highlightthickness=1,
58.         highlightcolor=COLORS['primary'],
59.         highlightbackground=COLORS['accent']
60.     )
61.     if show:
62.         entry.config(show=show)
63.     return entry
64. 
65. def create_label(parent, text, size=12, bold=False, fg=COLORS['text_light'],
    bg=COLORS['bg_dark']):
66.     """Create a modern styled label"""
67.     font_style = 'bold' if bold else 'normal'
68.     return tk.Label(
69.         parent,
70.         text=text,
71.         font=('Helvetica', size, font_style),
72.         fg=fg,
73.         bg=bg
74.     )
75. 
76. def create_frame(parent, padding_x=10, padding_y=10):
77.     """Create a modern styled frame"""
78.     return tk.Frame(
79.         parent,
80.         bg=COLORS['bg_dark'],
81.         padx=padding_x,
82.         pady=padding_y
83.     )
84. 
85. def _adjust_lightness(color, factor):
86.     """Adjust the lightness of a hex color"""
87.     # Simple lightness adjustment - not for production use
88.     r = int(color[1:3], 16)
```

```
    g = int(color[3:5], 16)
    b = int(color[5:7], 16)

    r = min(255, int(r * factor))
    g = min(255, int(g * factor))
    b = min(255, int(b * factor))

    return f'#{r:02x}{g:02x}{b:02x}'

class ATMGUI:
    def __init__(self, root):
                self.root = root
                self.root.title("ATM Simulator")
                self.atm = ATM()

                # Configure full screen
                self.root.attributes('-fullscreen', True)
                self.root.configure(bg=COLORS['bg_dark'])

                # Create main container with modern styling
                self.main_frame = create_frame(self.root, padding_x=30,
   padding_y=30)
                self.main_frame.pack(expand=True, fill='both')

                # Add ESC key binding to exit fullscreen
                self.root.bind('<Escape>', lambda e: self.root.attributes('-
   fullscreen', False))

                # Reference to balance display label
                self.balance_label = None

                # Create and show login frame
                self.create_login_frame()

          def create_login_frame(self):
                """Create the login screen with modern styling"""
                self.clear_frame()

                # Header container with logo effect
                header_frame = create_frame(self.main_frame)
                header_frame.pack(pady=(20, 40))

                # Title with modern styling
                title_label = create_label(header_frame, "BANK ATM TERMINAL",
   size=28, bold=True)
                title_label.pack()

                # Subtitle
```

```
134.             subtitle_label = create_label(header_frame, "Secure Banking
    Services", size=14, fg=COLORS['accent'])
135.             subtitle_label.pack(pady=(5, 0))
136.
137.             # Login container with card-like effect
138.             login_container = tk.Frame(
139.                 self.main_frame,
140.                 bg=COLORS['bg_medium'],
141.                 padx=40,
142.                 pady=40,
143.                 highlightbackground=COLORS['accent'],
144.                 highlightthickness=1
145.             )
146.             login_container.pack(padx=100, pady=10)
147.
148.             # Account Entry
149.             account_label = create_label(login_container, "Account Number:",
    bg=COLORS['bg_medium'])
150.             account_label.pack(anchor='w', pady=(0, 5))
151.
152.             self.account_entry = create_entry(login_container, width=20)
153.             self.account_entry.pack(pady=(0, 15), fill='x')
154.
155.             # PIN Entry
156.             pin_label = create_label(login_container, "Enter PIN:",
    bg=COLORS['bg_medium'])
157.             pin_label.pack(anchor='w', pady=(0, 5))
158.
159.             self.pin_entry = create_entry(login_container, show="•",
    width=20)
160.             self.pin_entry.pack(pady=(0, 25), fill='x')
161.
162.             # Button container
163.             button_frame = tk.Frame(login_container, bg=COLORS['bg_medium'])
164.             button_frame.pack(pady=(10, 0), fill='x')
165.
166.             # Login Button
167.             login_button = tk.Button(
168.                 button_frame,
169.                 text="LOGIN",
170.                 command=self.login,
171.                 font=('Helvetica', 12, 'bold'),
172.                 bg='white',
173.                 fg=COLORS['bg_dark'],
174.                 activebackground=COLORS['primary'],
175.                 activeforeground='white',
176.                 relief=tk.RAISED,
177.                 borderwidth=2,
```

```
                padx=10,
                pady=5,
                width=10,
                height=2,
                cursor="hand2"
            )
            login_button.pack(side=tk.LEFT, padx=(0, 10))
            login_button.bind("<Enter>", lambda e, c=COLORS['primary']:
   e.widget.config(bg=c, fg='white'))
            login_button.bind("<Leave>", lambda e:
   e.widget.config(bg='white', fg=COLORS['bg_dark']))

            # Register Button
            register_button = tk.Button(
                button_frame,
                text="REGISTER",
                command=self.show_registration_dialog,
                font=('Helvetica', 12, 'bold'),
                bg='white',
                fg=COLORS['bg_dark'],
                activebackground=COLORS['info'],
                activeforeground='white',
                relief=tk.RAISED,
                borderwidth=2,
                padx=10,
                pady=5,
                width=10,
                height=2,
                cursor="hand2"
            )
            register_button.pack(side=tk.LEFT, padx=10)
            register_button.bind("<Enter>", lambda e, c=COLORS['info']:
   e.widget.config(bg=c, fg='white'))
            register_button.bind("<Leave>", lambda e:
   e.widget.config(bg='white', fg=COLORS['bg_dark']))

            # Exit Button
            exit_button = tk.Button(
                button_frame,
                text="EXIT",
                command=self.root.quit,
                font=('Helvetica', 12, 'bold'),
                bg='white',
                fg=COLORS['bg_dark'],
                activebackground=COLORS['danger'],
                activeforeground='white',
                relief=tk.RAISED,
                borderwidth=2,
```

```
                padx=10,
                pady=5,
                width=10,
                height=2,
                cursor="hand2"
            )
            exit_button.pack(side=tk.LEFT, padx=10)
            exit_button.bind("<Enter>", lambda e, c=COLORS['danger']:
    e.widget.config(bg=c, fg='white'))
            exit_button.bind("<Leave>", lambda e: e.widget.config(bg='white',
    fg=COLORS['bg_dark']))

            # Footer
            footer_frame = create_frame(self.main_frame)
            footer_frame.pack(side=tk.BOTTOM, fill='x', pady=20)

            footer_text = create_label(
                footer_frame,
                "© 2025 Modern Banking ATM • Made by Charu , Nandini , Anuj &
    Aashutosh",
                size=10,
                fg=COLORS['accent']
            )
            footer_text.pack(side=tk.RIGHT, padx=20)

            # Focus on account entry
            self.account_entry.focus_set()

        def create_menu_frame(self):
            """Create the main menu screen with modern styling"""
            self.clear_frame()

            # Header with customer info
            header_frame = create_frame(self.main_frame)
            header_frame.pack(fill='x', pady=(0, 30))

            # Add current time indicator (just for visual effect)
            time_label = create_label(header_frame, "Session Active",
    size=10, fg=COLORS['accent'])
            time_label.pack(side=tk.RIGHT, padx=10)

            # Customer welcome
            customer_name = self.atm.get_customer_name()
            name_label = create_label(header_frame, f"Welcome,
    {customer_name}", size=16, bold=True)
            name_label.pack(side=tk.LEFT, padx=10)

            # Balance display
```

```
            balance_frame = tk.Frame(
                self.main_frame,
                bg=COLORS['bg_medium'],
                padx=20,
                pady=15
            )
            balance_frame.pack(fill='x', pady=(0, 30))

            balance_label = create_label(
                balance_frame,
                "Current Balance",
                size=12,
                fg=COLORS['accent'],
                bg=COLORS['bg_medium']
            )
            balance_label.pack()

            # Store reference to the balance amount label for updates
            self.balance_label = create_label(
                balance_frame,
                f"\u20B9 {self.atm.check_balance():.2f}",
                size=24,
                bold=True,
                bg=COLORS['bg_medium']
            )
            self.balance_label.pack()

            # Menu container
            menu_container = create_frame(self.main_frame)
            menu_container.pack(expand=True, fill='both')

            # Create a grid for menu options
            menu_items = [
                ("Deposit", self.deposit, COLORS['success']),
                ("Withdraw", self.withdraw, COLORS['primary']),
                ("Transaction History", self.show_history, COLORS['info']),
                ("Change PIN", self.change_pin, COLORS['warning']),
                ("Logout", self.logout, COLORS['danger']),
                ("Exit ATM", self.root.quit, COLORS['muted'])
            ]

            # Create card-like containers for each menu option
            for i, (text, command, color) in enumerate(menu_items):
                row, col = divmod(i, 3)

                # Create a card-like container
                card_frame = tk.Frame(
                    menu_container,
```

```
313.                     bg=COLORS['bg_medium'],
314.                     padx=20,
315.                     pady=20,
316.                     highlightbackground=color,
317.                     highlightthickness=2
318.                 )
319.                 card_frame.grid(row=row, column=col, padx=15, pady=15,
    sticky='nsew')
320. 
321.                 # Create card content container for better organization
322.                 content_frame = tk.Frame(card_frame, bg=COLORS['bg_medium'])
323.                 content_frame.pack(fill='both', expand=True)
324. 
325.                 # Header container
326.                 header_frame = tk.Frame(content_frame,
    bg=COLORS['bg_medium'])
327.                 header_frame.pack(fill='x', anchor='nw')
328. 
329.                 # Add icon placeholder (simple colored square for now)
330.                 icon = tk.Frame(header_frame, bg=color, width=30, height=30)
331.                 icon.pack(side=tk.LEFT, pady=(0, 10))
332. 
333.                 # Option name - place next to icon
334.                 option_name = create_label(header_frame, text, size=16,
    bold=True, bg=COLORS['bg_medium'])
335.                 option_name.pack(side=tk.LEFT, padx=(10, 0), pady=(0, 10))
336. 
337.                 # Make the entire card clickable - add a button that fills
    most of the space
338.                 action_button = tk.Button(
339.                     content_frame,
340.                     text="SELECT",
341.                     command=command,
342.                     font=('Helvetica', 12, 'bold'),
343.                     bg='white',  # High contrast button
344.                     fg=COLORS['bg_dark'],  # Dark text for contrast
345.                     activebackground=color,
346.                     activeforeground='white',
347.                     relief=tk.RAISED,  # Give it a raised appearance
348.                     borderwidth=2,
349.                     padx=10,
350.                     pady=5,
351.                     width=15,  # Make button wider
352.                     height=2,   # Make button taller
353.                     cursor="hand2"  # Change cursor to hand when hovering
354.                 )
355.                 action_button.pack(side=tk.BOTTOM, fill='x', pady=(15, 0))
356. 
```

```
                # Add hover effect to the button
                action_button.bind("<Enter>", lambda e, c=color:
    e.widget.config(bg=c, fg='white'))
                action_button.bind("<Leave>", lambda e:
    e.widget.config(bg='white', fg=COLORS['bg_dark']))

                # Make the entire card feel clickable by changing cursor on
    hover
                for widget in [card_frame, content_frame, header_frame,
    option_name, icon]:
                    widget.bind("<Enter>", lambda e, b=action_button,
    c=color: b.config(bg=c, fg='white'))
                    widget.bind("<Leave>", lambda e, b=action_button:
    b.config(bg='white', fg=COLORS['bg_dark']))

            # Configure grid to be responsive
            for i in range(3):
                menu_container.columnconfigure(i, weight=1)
            menu_container.rowconfigure(0, weight=1)
            menu_container.rowconfigure(1, weight=1)

        def clear_frame(self):
            """Clear all widgets from the main frame"""
            for widget in self.main_frame.winfo_children():
                widget.destroy()

        def login(self):
            """Handle login with full account number"""
            full_account = self.account_entry.get()
            pin = self.pin_entry.get()

            success, message = self.atm.login(full_account, pin)
            if success:
                self.create_menu_frame()
            else:
                if "Please register" in message:
                    response = messagebox.askyesno("Account Not Found",
                        f"{message}\nWould you like to register a new
    account?")
                    if response:
                        self.show_registration_dialog(full_account)
                else:
                    messagebox.showerror("Login Failed", message)
                    if "locked" in message:
                        self.root.quit()

        def show_registration_dialog(self, account_number=""):
            """Show registration dialog for new users with modern styling"""
```

```
398.                 dialog = tk.Toplevel(self.root)
399.                 dialog.title("New Account Registration")
400.                 dialog.geometry("450x520")
401.                 dialog.configure(bg=COLORS['bg_dark'])
402.
403.                 # Add some padding
404.                 container = create_frame(dialog, padding_x=25, padding_y=25)
405.                 container.pack(fill='both', expand=True)
406.
407.                 # Title
408.                 title = create_label(container, "Create New Account", size=18,
    bold=True)
409.                 title.pack(pady=(0, 20))
410.
411.                 # Form fields
412.                 # Account Number
413.                 account_label = create_label(container, "Account Number:")
414.                 account_label.pack(anchor='w', pady=(0, 5))
415.
416.                 account_entry = create_entry(container)
417.                 account_entry.pack(fill='x', pady=(0, 15))
418.                 if account_number:
419.                     account_entry.insert(0, account_number)
420.                     account_entry.config(state='readonly')
421.
422.                 # Name
423.                 name_label = create_label(container, "Full Name:")
424.                 name_label.pack(anchor='w', pady=(0, 5))
425.
426.                 name_entry = create_entry(container)
427.                 name_entry.pack(fill='x', pady=(0, 15))
428.
429.                 # PIN
430.                 pin_label = create_label(container, "Create 4-digit PIN:")
431.                 pin_label.pack(anchor='w', pady=(0, 5))
432.
433.                 pin_entry = create_entry(container, show="•")
434.                 pin_entry.pack(fill='x', pady=(0, 15))
435.
436.                 # Initial Deposit
437.                 deposit_label = create_label(container, "Initial Deposit:")
438.                 deposit_label.pack(anchor='w', pady=(0, 5))
439.
440.                 deposit_entry = create_entry(container)
441.                 deposit_entry.pack(fill='x', pady=(0, 15))
442.                 deposit_entry.insert(0, "0")
443.
444.                 # Status message
```

```
            status_label = create_label(container, "", fg=COLORS['warning'])
            status_label.pack(pady=10)

            def register():
                full_account = account_entry.get()
                name = name_entry.get()
                pin = pin_entry.get()

                try:
                    deposit = float(deposit_entry.get())
                    if deposit < 0:
                        raise ValueError
                except ValueError:
                    status_label.config(text="Invalid deposit amount")
                    return

                # Check if account already exists
                if full_account in self.atm.accounts:
                    status_label.config(text="Account already registered!")
                    return

                if not name.strip():
                    status_label.config(text="Name cannot be empty")
                    return

                if len(pin) != 4 or not pin.isdigit():
                    status_label.config(text="PIN must be 4 digits")
                    return

                success, message = self.atm.register_user(
                    full_account, name, pin, deposit
                )

                if success:
                    messagebox.showinfo("Success", message)
                    dialog.destroy()
                    # Auto-login the new user
                    self.atm.login(full_account, pin)
                    self.create_menu_frame()
                else:
                    status_label.config(text=message)

            # Button container
            button_frame = create_frame(container, padding_x=0, padding_y=0)
            button_frame.pack(pady=10)

            # Register button
            register_button = tk.Button(
```

```
                    button_frame,
                    text="REGISTER",
                    command=register,
                    font=('Helvetica', 12, 'bold'),
                    bg='white',
                    fg=COLORS['bg_dark'],
                    activebackground=COLORS['success'],
                    activeforeground='white',
                    relief=tk.RAISED,
                    borderwidth=2,
                    padx=10,
                    pady=5,
                    width=10,
                    height=2,
                    cursor="hand2"
                )
                register_button.pack(side=tk.LEFT, padx=10)
                register_button.bind("<Enter>", lambda e, c=COLORS['success']:
    e.widget.config(bg=c, fg='white'))
                register_button.bind("<Leave>", lambda e:
    e.widget.config(bg='white', fg=COLORS['bg_dark']))

                # Cancel button
                cancel_button = tk.Button(
                    button_frame,
                    text="CANCEL",
                    command=dialog.destroy,
                    font=('Helvetica', 12, 'bold'),
                    bg='white',
                    fg=COLORS['bg_dark'],
                    activebackground=COLORS['danger'],
                    activeforeground='white',
                    relief=tk.RAISED,
                    borderwidth=2,
                    padx=10,
                    pady=5,
                    width=10,
                    height=2,
                    cursor="hand2"
                )
                cancel_button.pack(side=tk.LEFT, padx=10)
                cancel_button.bind("<Enter>", lambda e, c=COLORS['danger']:
    e.widget.config(bg=c, fg='white'))
                cancel_button.bind("<Leave>", lambda e:
    e.widget.config(bg='white', fg=COLORS['bg_dark']))

            def logout(self):
                """Handle logout"""
```

```
537.                 self.atm.logout()
538.                 self.create_login_frame()
539.
540.             def check_balance(self):
541.                 """Show current balance"""
542.                 balance = self.atm.check_balance()
543.                 messagebox.showinfo("Account Balance",
544.                                     f"Current Balance: \u20B9 {balance:.2f}")
545.
546.             def update_balance_display(self):
547.                 """Update the balance display in the main menu"""
548.                 if hasattr(self, 'balance_label') and self.balance_label:
549.                     self.balance_label.config(text=f"\u20B9
    {self.atm.check_balance():.2f}")
550.
551.             def deposit(self):
552.                 """Handle deposit"""
553.                 amount = self.get_amount("Enter deposit amount:")
554.                 if amount is not None:
555.                     if self.atm.deposit(amount):
556.                         messagebox.showinfo("Deposit Successful",
557.                                             f"\u20B9 {amount:.2f} deposited
    successfully.\nNew Balance: \u20B9 {self.atm.check_balance():.2f}")
558.                         # Update the balance display
559.                         self.update_balance_display()
560.                     else:
561.                         messagebox.showerror("Error", "Invalid deposit amount!")
562.
563.             def withdraw(self):
564.                 """Handle withdrawal"""
565.                 amount = self.get_amount("Enter withdrawal amount:")
566.                 if amount is not None:
567.                     if self.atm.withdraw(amount):
568.                         messagebox.showinfo("Withdrawal Successful",
569.                                             f"\u20B9 {amount:.2f} withdrawn
    successfully.\nNew Balance: \u20B9 {self.atm.check_balance():.2f}")
570.                         # Update the balance display
571.                         self.update_balance_display()
572.                     else:
573.                         messagebox.showerror("Error", "Invalid amount or
    insufficient funds!")
574.
575.             def change_pin(self):
576.                 """Handle PIN change with modern styling"""
577.                 dialog = tk.Toplevel(self.root)
578.                 dialog.title("Change PIN")
579.                 dialog.geometry("400x350")
580.                 dialog.configure(bg=COLORS['bg_dark'])
```

```
581. 
582.             # Container
583.             container = create_frame(dialog, padding_x=25, padding_y=25)
584.             container.pack(fill='both', expand=True)
585. 
586.             # Title
587.             title = create_label(container, "Change Your PIN", size=18,
    bold=True)
588.             title.pack(pady=(0, 20))
589. 
590.             # Old PIN
591.             old_pin_label = create_label(container, "Current PIN:")
592.             old_pin_label.pack(anchor='w', pady=(0, 5))
593. 
594.             old_pin_entry = create_entry(container, show="•")
595.             old_pin_entry.pack(fill='x', pady=(0, 15))
596. 
597.             # New PIN
598.             new_pin_label = create_label(container, "New 4-digit PIN:")
599.             new_pin_label.pack(anchor='w', pady=(0, 5))
600. 
601.             new_pin_entry = create_entry(container, show="•")
602.             new_pin_entry.pack(fill='x', pady=(0, 15))
603. 
604.             # Confirm New PIN
605.             confirm_pin_label = create_label(container, "Confirm New PIN:")
606.             confirm_pin_label.pack(anchor='w', pady=(0, 5))
607. 
608.             confirm_pin_entry = create_entry(container, show="•")
609.             confirm_pin_entry.pack(fill='x', pady=(0, 15))
610. 
611.             def confirm():
612.                 old_pin = old_pin_entry.get()
613.                 new_pin = new_pin_entry.get()
614.                 confirm_pin = confirm_pin_entry.get()
615. 
616.                 if new_pin != confirm_pin:
617.                     messagebox.showerror("Error", "New PINs don't match!")
618.                     return
619. 
620.                 if self.atm.change_pin(old_pin, new_pin):
621.                     messagebox.showinfo("Success", "PIN changed
    successfully!")
622.                     dialog.destroy()
623.                 else:
624.                     messagebox.showerror("Error", "Invalid current PIN or new
    PIN must be 4 digits!")
625. 
```

```
626.             # Confirm button
627.             confirm_button = tk.Button(
628.                 container,
629.                 text="CONFIRM",
630.                 command=confirm,
631.                 font=('Helvetica', 12, 'bold'),
632.                 bg='white',
633.                 fg=COLORS['bg_dark'],
634.                 activebackground=COLORS['success'],
635.                 activeforeground='white',
636.                 relief=tk.RAISED,
637.                 borderwidth=2,
638.                 padx=10,
639.                 pady=5,
640.                 width=10,
641.                 height=2,
642.                 cursor="hand2"
643.             )
644.             confirm_button.pack(pady=20)
645.             confirm_button.bind("<Enter>", lambda e, c=COLORS['success']:
    e.widget.config(bg=c, fg='white'))
646.             confirm_button.bind("<Leave>", lambda e:
    e.widget.config(bg='white', fg=COLORS['bg_dark']))
647.
648.         def show_history(self):
649.             """Show transaction history with modern styling"""
650.             history = self.atm.get_transaction_history()
651.             if not history:
652.                 messagebox.showinfo("Transaction History", "No transactions
    found")
653.                 return
654.
655.             dialog = tk.Toplevel(self.root)
656.             dialog.title("Transaction History")
657.             dialog.geometry("500x400")
658.             dialog.configure(bg=COLORS['bg_dark'])
659.
660.             # Container
661.             container = create_frame(dialog, padding_x=25, padding_y=25)
662.             container.pack(fill='both', expand=True)
663.
664.             # Title
665.             title = create_label(container, "Recent Transactions", size=18,
    bold=True)
666.             title.pack(pady=(0, 20))
667.
668.             # Create custom style for treeview
669.             style = ttk.Style()
```

```
670.                style.theme_use('default')
671.                style.configure('Treeview',
672.                                background=COLORS['bg_medium'],
673.                                foreground=COLORS['text_light'],
674.                                rowheight=25,
675.                                fieldbackground=COLORS['bg_medium'])
676.                style.configure('Treeview.Heading',
677.                                background=COLORS['accent'],
678.                                foreground=COLORS['text_light'],
679.                                font=('Helvetica', 10, 'bold'))
680.                style.map('Treeview', background=[('selected',
    COLORS['primary'])])
681.
682.                # Transaction list
683.                tree_frame = tk.Frame(container, bg=COLORS['bg_dark'])
684.                tree_frame.pack(fill='both', expand=True)
685.
686.                tree = ttk.Treeview(tree_frame, columns=('Date', 'Transaction'),
    show='headings')
687.                tree.heading('Date', text='Date')
688.                tree.heading('Transaction', text='Transaction')
689.                tree.column('Date', width=150)
690.                tree.column('Transaction', width=350)
691.
692.                for i, transaction in enumerate(reversed(history)):
693.                    # Add date as first column (placeholder)
694.                    parts = transaction.split(' - ')
695.                    date = parts[0] if len(parts) > 1 else "Today"
696.                    details = parts[1] if len(parts) > 1 else transaction
697.                    tree.insert('', 'end', values=(date, details))
698.
699.                scrollbar = ttk.Scrollbar(tree_frame, orient="vertical",
    command=tree.yview)
700.                tree.configure(yscrollcommand=scrollbar.set)
701.
702.                tree.pack(side='left', fill='both', expand=True)
703.                scrollbar.pack(side='right', fill='y')
704.
705.                # Close button
706.                close_button = tk.Button(
707.                    container,
708.                    text="CLOSE",
709.                    command=dialog.destroy,
710.                    font=('Helvetica', 12, 'bold'),
711.                    bg='white',
712.                    fg=COLORS['bg_dark'],
713.                    activebackground=COLORS['accent'],
714.                    activeforeground='white',
```

```
                    relief=tk.RAISED,
                    borderwidth=2,
                    padx=10,
                    pady=5,
                    width=10,
                    height=2,
                    cursor="hand2"
                )
                close_button.pack(pady=(20, 0))
                close_button.bind("<Enter>", lambda e, c=COLORS['accent']:
    e.widget.config(bg=c, fg='white'))
                close_button.bind("<Leave>", lambda e:
    e.widget.config(bg='white', fg=COLORS['bg_dark']))

            def get_amount(self, prompt):
                """Get amount from user with modern styling"""
                dialog = tk.Toplevel(self.root)
                dialog.title("Amount Entry")
                dialog.geometry("400x250")
                dialog.configure(bg=COLORS['bg_dark'])

                # Container
                container = create_frame(dialog, padding_x=25, padding_y=25)
                container.pack(fill='both', expand=True)

                # Prompt
                prompt_label = create_label(container, prompt, size=14)
                prompt_label.pack(pady=15)

                # Amount entry
                amount_entry = create_entry(container, width=20)
                amount_entry.pack(pady=15)
                amount_entry.focus_set()

                amount = None

                def confirm():
                    nonlocal amount
                    try:
                        amount = float(amount_entry.get())
                        if amount <= 0:
                            raise ValueError
                        dialog.destroy()
                    except ValueError:
                        messagebox.showerror("Error", "Please enter a valid
    positive amount!")

                # Button container
```

```
760.                button_frame = create_frame(container, padding_x=0, padding_y=0)
761.                button_frame.pack(pady=10)
762.
763.                # Confirm button
764.                confirm_button = tk.Button(
765.                    button_frame,
766.                    text="CONFIRM",
767.                    command=confirm,
768.                    font=('Helvetica', 12, 'bold'),
769.                    bg='white',
770.                    fg=COLORS['bg_dark'],
771.                    activebackground=COLORS['success'],
772.                    activeforeground='white',
773.                    relief=tk.RAISED,
774.                    borderwidth=2,
775.                    padx=10,
776.                    pady=5,
777.                    width=10,
778.                    height=2,
779.                    cursor="hand2"
780.                )
781.                confirm_button.pack(side=tk.LEFT, padx=10)
782.                confirm_button.bind("<Enter>", lambda e, c=COLORS['success']:
    e.widget.config(bg=c, fg='white'))
783.                confirm_button.bind("<Leave>", lambda e:
    e.widget.config(bg='white', fg=COLORS['bg_dark']))
784.
785.                # Cancel button
786.                cancel_button = tk.Button(
787.                    button_frame,
788.                    text="CANCEL",
789.                    command=dialog.destroy,
790.                    font=('Helvetica', 12, 'bold'),
791.                    bg='white',
792.                    fg=COLORS['bg_dark'],
793.                    activebackground=COLORS['danger'],
794.                    activeforeground='white',
795.                    relief=tk.RAISED,
796.                    borderwidth=2,
797.                    padx=10,
798.                    pady=5,
799.                    width=10,
800.                    height=2,
801.                    cursor="hand2"
802.                )
803.                cancel_button.pack(side=tk.LEFT, padx=10)
804.                cancel_button.bind("<Enter>", lambda e, c=COLORS['danger']:
    e.widget.config(bg=c, fg='white'))
```

```
805.             cancel_button.bind("<Leave>", lambda e:
    e.widget.config(bg='white', fg=COLORS['bg_dark']))
806.
807.             # Make dialog modal
808.             dialog.transient(self.root)
809.             dialog.grab_set()
810.             self.root.wait_window(dialog)
811.
812.             return amount
813.
814.     if __name__ == "__main__":
815.         root = tk.Tk()
816.         app = ATMGUI(root)
817.         root.mainloop()
```

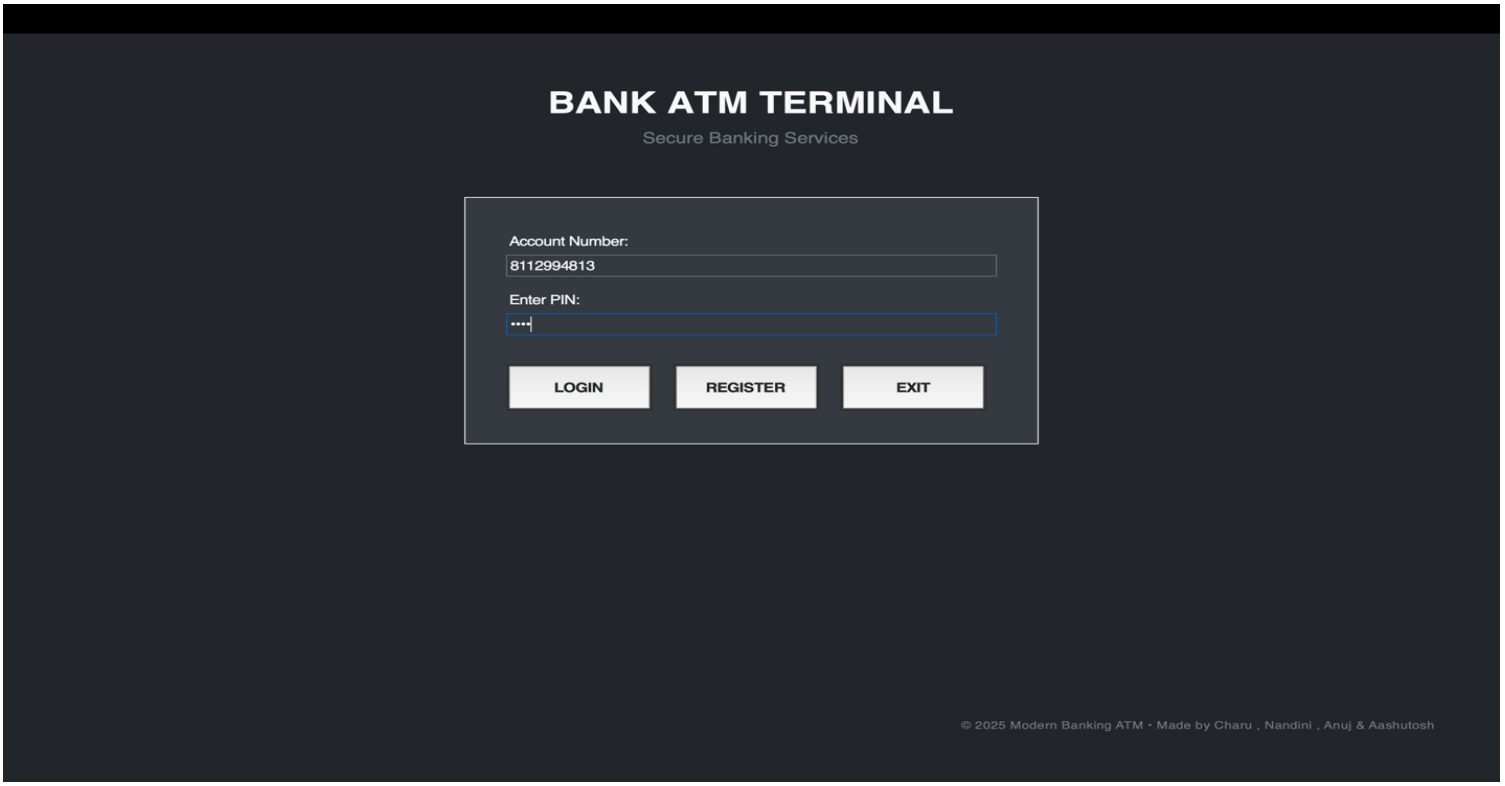
## 2. atm.py (Core Business Logic):

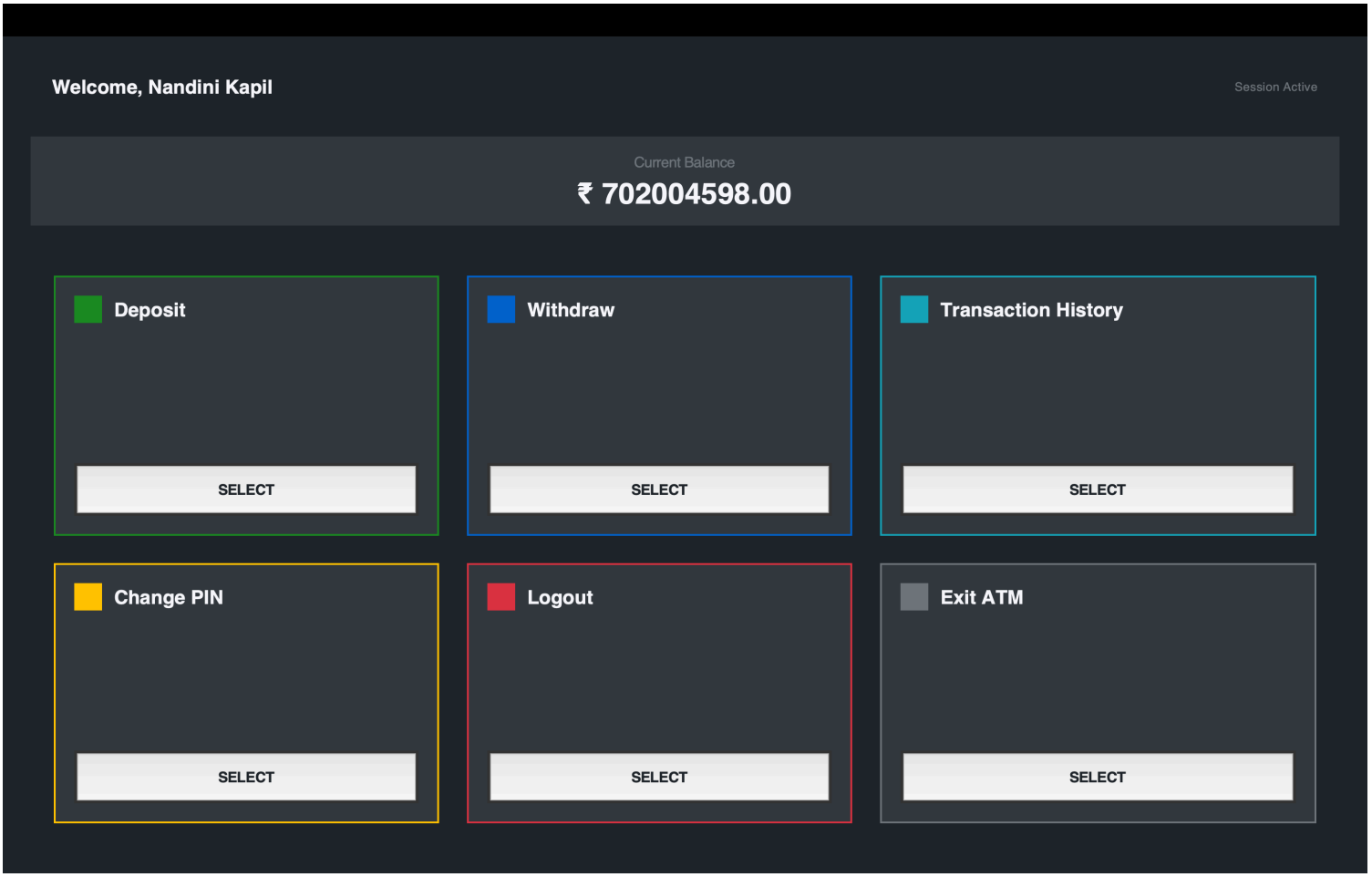
```
import json
from datetime import datetime
import os

class ATM:
    def __init__(self):
        self.current_account = None
        self.pin_attempts = 0
        self.max_pin_attempts = 3
        self.accounts_file = "users.json"
        self.accounts = self._load_accounts()

    def _load_accounts(self):
        """Load accounts from JSON file or create default if not exists"""
        if os.path.exists(self.accounts_file):
            try:
                with open(self.accounts_file, 'r') as f:
                    # Check if file is empty
                    if os.stat(self.accounts_file).st_size == 0:
                        return self._create_default_accounts()
                    return json.load(f)
            except json.JSONDecodeError:
                # If JSON is invalid, create default accounts
                return self._create_default_accounts()
        else:
            return self._create_default_accounts()

    def _create_default_accounts(self):
        """Create default accounts and save to file"""
        default_accounts = {
            "10001234": {
```

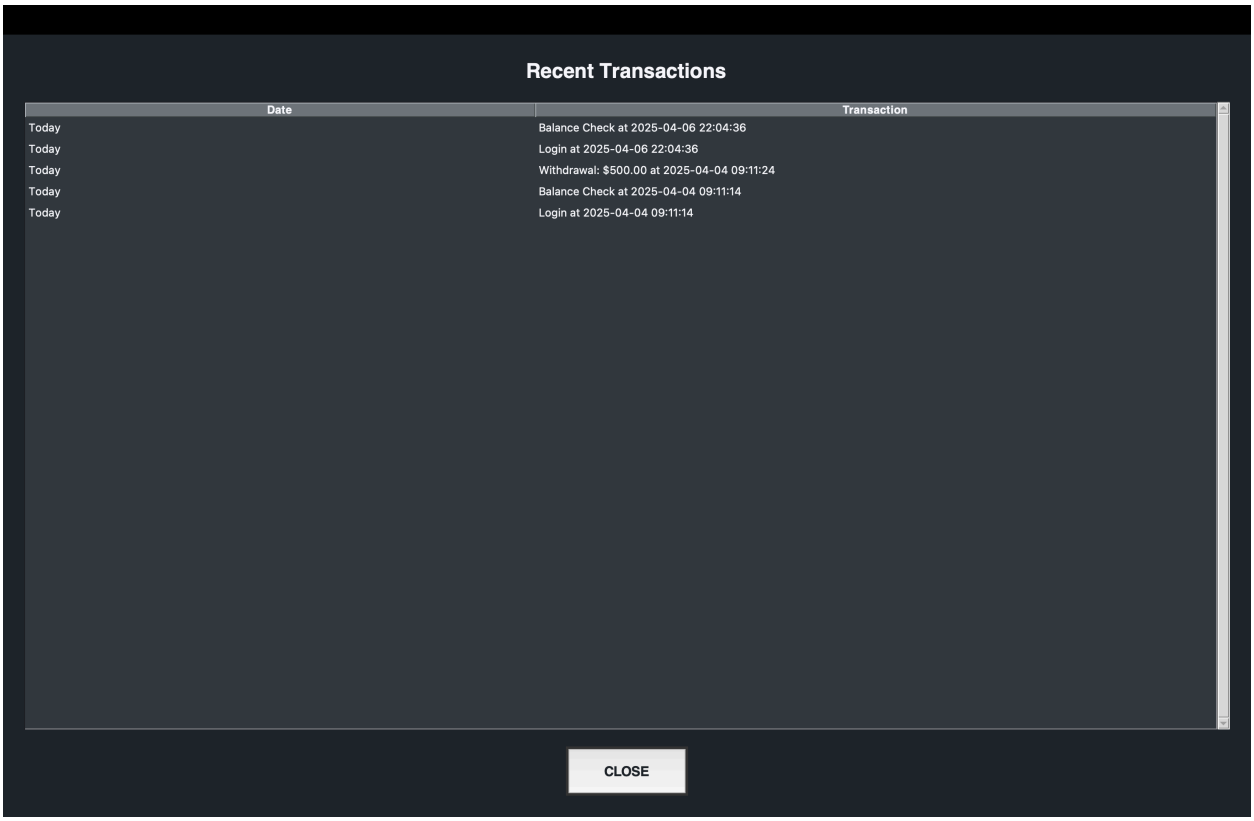
```
                "pin": "1234",
                "balance": 1000.0,
                "name": "John Doe",
                "transaction_history": []
            },
            "20005678": {
                "pin": "5678",
                "balance": 2500.0,
                "name": "Jane Smith",
                "transaction_history": []
            }
        }
        self._save_accounts(default_accounts)
        return default_accounts

    def _save_accounts(self, accounts=None):
        """Save accounts to JSON file"""
        with open(self.accounts_file, 'w') as f:
            json.dump(accounts or self.accounts, f, indent=4)

    def get_all_users(self):
        """Return all registered users"""
        return self.accounts

    def register_user(self, full_account, name, pin, initial_deposit=0):
        """Register a new user"""
        if full_account in self.accounts:
            return False, "Account already exists"

        if len(pin) != 4 or not pin.isdigit():
            return False, "PIN must be 4 digits"

        if initial_deposit < 0:
            return False, "Initial deposit cannot be negative"

        self.accounts[full_account] = {
            "pin": pin,
            "balance": float(initial_deposit),
            "name": name,
            "transaction_history": [f"Account created with initial deposit:
${initial_deposit:.2f}"]
        }

        self._save_accounts()
        return True, "Registration successful"

    def login(self, full_account, pin):
        """Authenticate user with full account number and PIN"""
```

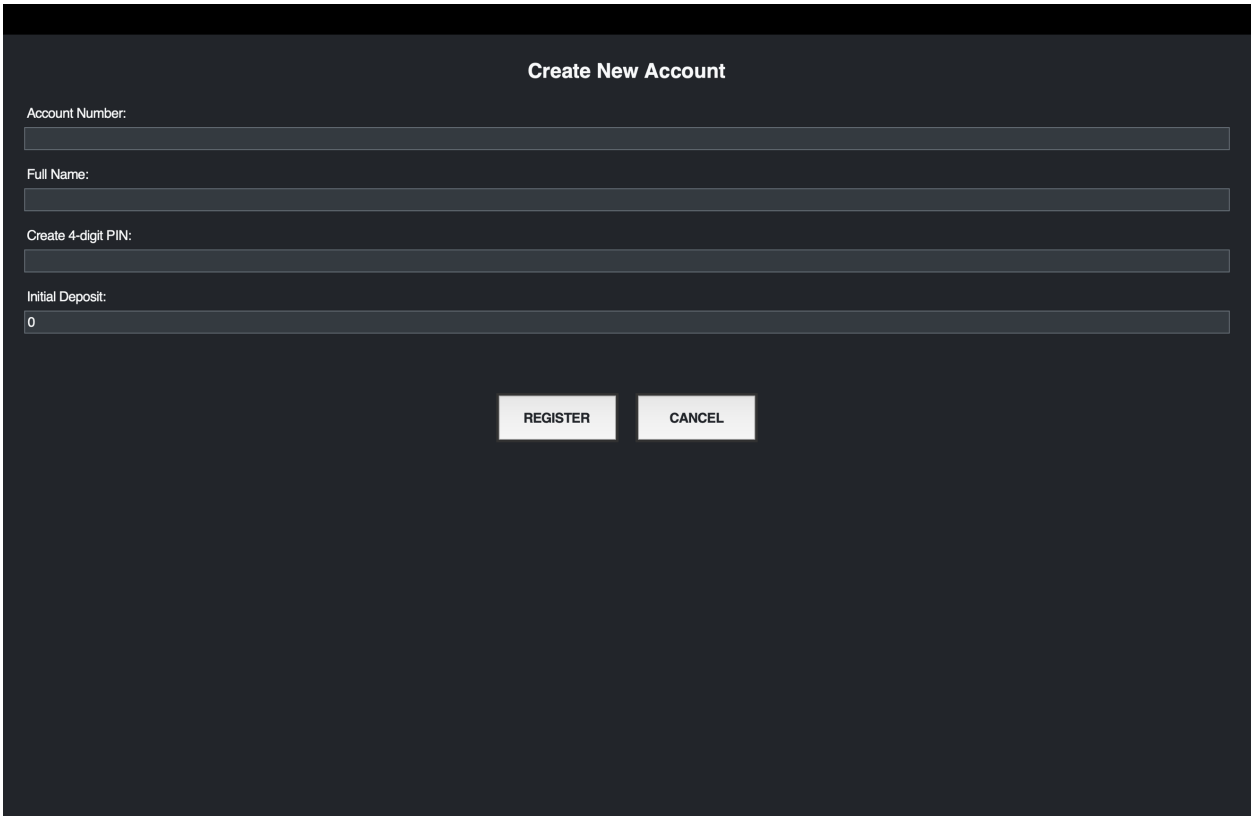
```
        if self.pin_attempts >= self.max_pin_attempts:
            return False, "Too many attempts. Account locked."

        if full_account in self.accounts:
            if pin == self.accounts[full_account]["pin"]:
                self.current_account = full_account
                self.pin_attempts = 0
                self._add_transaction("Login")
                return True, "Login successful"
            else:
                self.pin_attempts += 1
                remaining = self.max_pin_attempts - self.pin_attempts
                return False, f"Invalid PIN. {remaining} attempts remaining."
        return False, "Account not found. Please register."

    def logout(self):
        """Logout the current user"""
        if self.current_account:
            self._add_transaction("Logout")
        self.current_account = None
        return True

    def check_balance(self):
        """Return current balance"""
        if self.current_account:
            balance = self.accounts[self.current_account]["balance"]
            self._add_transaction("Balance Check")
            return balance
        return None

    def deposit(self, amount):
        """Deposit money into account"""
        if self.current_account and amount > 0:
            self.accounts[self.current_account]["balance"] += amount
            self._add_transaction(f"Deposit: ${amount:.2f}")
            self._save_accounts()
            return True
        return False

    def withdraw(self, amount):
        """Withdraw money from account"""
        if (self.current_account and amount > 0 and
            amount <= self.accounts[self.current_account]["balance"]):
            self.accounts[self.current_account]["balance"] -= amount
            self._add_transaction(f"Withdrawal: ${amount:.2f}")
            self._save_accounts()
            return True
        return False
```

```
    def change_pin(self, old_pin, new_pin):
        """Change PIN if old PIN is correct"""
        if (self.current_account and
            old_pin == self.accounts[self.current_account]["pin"] and
            len(new_pin) == 4 and new_pin.isdigit()):
            self.accounts[self.current_account]["pin"] = new_pin
            self._add_transaction("PIN Changed")
            self._save_accounts()
            return True
        return False

    def get_transaction_history(self):
        """Get last 5 transactions"""
        if self.current_account:
            return self.accounts[self.current_account]["transaction_history"][-5:]
        return []

    def get_customer_name(self):
        """Get current customer name"""
        if self.current_account:
            return self.accounts[self.current_account]["name"]
        return ""

    def _add_transaction(self, transaction_type):
        """Add transaction to history"""
        if self.current_account:
            self.accounts[self.current_account]["transaction_history"].append(
                f"{transaction_type} at {self._get_current_time()}"
            )

    def _get_current_time(self):
        """Helper to get current time string"""
        return datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    def is_authenticated(self):
        """Check if user is logged in"""
        return self.current_account is not None
```


BANK ATM TERMINAL
Secure Banking Services
Account Number:
8112994813
Enter PIN:
LOGIN
REGISTER
EXIT
© 2025 Modern Banking ATM • Made by Charu , Nandini , Anuj & Aashutosh


Welcome, Nandini Kapil
Session Active
Current Balance
₹ 702004598.00
Deposit
SELECT
Withdraw
SELECT
Transaction History
SELECT
Change PIN
SELECT
Logout
SELECT
Exit ATM
SELECT


Recent Transactions
Date
Transaction
Today
Balance Check at 2025-04-06 22:04:36
Today
Login at 2025-04-06 22:04:36
Today
Withdrawal: $500.00 at 2025-04-04 09:11:24
Today
Balance Check at 2025-04-04 09:11:14
Today
Login at 2025-04-04 09:11:14
CLOSE


Create New Account
Account Number:
Full Name:
Create 4-digit PIN:
Initial Deposit:
0
REGISTER
CANCEL

## Sample Input /Output

**Sample Input 1: Login Process**

- Account Number: 10001234
- PIN: 1234

**Sample Output 1:**

- Welcome screen displays with user name "Nandini Kapil"
- Account balance shows ₹1000.00
- Transaction options are enabled

**Sample Input 2: Deposit Process**

- Amount: ₹500

**Sample Output 2:**

- Transaction successful message
- Updated balance shows ₹1500.00
- Transaction is recorded in history

**Sample Input 3: Withdrawal Process**

- Amount: ₹ 200

**Sample Output 3:**

- Transaction successful message
- Updated balance shows ₹1300.00
- Transaction is recorded in history

## Test Case Descriptions

### 1. Test Case ID: TC001

**Description:** Validate user login with correct credentials.**Preconditions:** User should be registered.**Test Steps:**

- Open the application.
- Enter valid account number (10001234) and PIN (1234).
- Click on the login button.

**Expected Result:** User is successfully logged in.**Actual Result:** User logged in successfully.
**Status:** Pass

### 2. Test Case ID: TC002

**Description:** Validate user login with incorrect PIN.**Preconditions:** User should be registered.**Test Steps:**

- Open the application.
- Enter valid account number (10001234) but incorrect PIN (1111).
- Click on the login button.

**Expected Result:** Error message indicating incorrect PIN with remaining attempts.**Actual Result:** System displays "Invalid PIN. 2 attempts remaining."
**Status:** Pass

**3. Test Case ID: TC003**
**Description:** Validate deposit functionality.**Preconditions:** User should be logged in.**Test Steps:**

- Navigate to the deposit screen.
- Enter amount to deposit ($500).
- Click on confirm button.

**Expected Result:** Amount should be added to the balance and transaction history updated.**Actual Result:** Amount added successfully, balance updated to reflect the change, transaction recorded in history.
**Status:** Pass

**4. Test Case ID: TC004**
**Description:** Validate withdrawal functionality with sufficient funds.**Preconditions:** User should be logged in with sufficient balance.**Test Steps:**

- Navigate to the withdrawal screen.
- Enter amount to withdraw ($200).
- Click on confirm button.

**Expected Result:** Amount should be deducted from the balance and transaction history updated.**Actual Result:** Amount deducted successfully, balance updated to reflect the change, transaction recorded in history.
**Status:** Pass

**5. Test Case ID: TC005**
**Description:** Validate withdrawal functionality with insufficient funds.**Preconditions:** User should be logged in.**Test Steps:**

- Navigate to the withdrawal screen.
- Enter amount to withdraw greater than current balance (e.g., $5000 with $1000 balance).
- Click on confirm button.

**Expected Result:** Error message indicating insufficient funds.**Actual Result:** System displays "Insufficient funds for this transaction."
**Status:** Pass

**6. Test Case ID: TC006**
**Description:** Validate new user registration.**Preconditions:** Registration screen is accessible.**Test Steps:**

- Click on the "Register" button from the login screen.
- Enter new account details (Name: "New User", Account: 30007890, PIN: 5555, Initial Deposit: $500).
- Click on register button.

**Expected Result:** New account should be created and stored in the system.**Actual Result:** Account created successfully with initial deposit.
**Status:** Pass

**7. Test Case ID: TC007**
**Description:** Validate PIN change functionality.**Preconditions:** User should be logged in.**Test Steps:**

- Navigate to the PIN change screen.
- Enter current PIN (1234).
- Enter new PIN (4321).
- Confirm new PIN (4321).
- Click on change PIN button.

**Expected Result:** PIN should be updated successfully.**Actual Result:** PIN updated in the system, confirmation message displayed.
**Status:** Pass

**8. Test Case ID: TC008**
**Description:** Validate transaction history display.**Preconditions:** User should be logged in with some transaction history.**Test Steps:**

- Navigate to the transaction history screen.

**Expected Result:** Transaction history should display recent transactions with timestamps.**Actual Result:** Transaction history displayed with proper formatting and timestamps.
**Status:** Pass

# Conclusion:

**Summary of test outcomes**

The Banking ATM Simulator successfully passed all test cases, demonstrating robust functionality across all core banking operations including login authentication, deposit and withdrawal processing, account creation, PIN management, and transaction history tracking. The system correctly handled both valid and invalid inputs, providing appropriate feedback to users in all scenarios.

**Met objective**

- Successfully implemented a fully functional ATM simulator with modern GUI
- Created secure user authentication using PIN-based login
- Implemented core banking operations (deposit, withdrawal, balance inquiry)
- Developed transaction history tracking with timestamps
- Created an intuitive, user-friendly interface
- Implemented robust data persistence using JSON storage
- Ensured proper input validation and error handling

**Further improved required**

- Implement encryption for PIN storage to enhance security
- Add multi-factor authentication options
- Develop additional banking features such as fund transfers between accounts
- Implement a more robust database solution for larger scale deployments

- Add internationalization support for multiple languages
- Create a responsive design for various screen sizes
- Implement automated logging for system monitoring
- Develop a comprehensive backup and recovery system