Bank of Python: Detailed Project Documentation (with Logging)

This document offers a **comprehensive reference** for students, covering all modules and how each integrates with the newly introduced **logging system**. The goal is to help learners see the **end-to-end** structure of the banking application, from onboarding to transactions, with a focus on **robust logging** for traceability.

---

1. Introduction

Bank of Python is an educational project that simulates basic banking services. By reviewing this document, students will understand:

1. **Application Architecture**: How modules are organized to handle onboarding, user authentication, operations, and services.
2. **New Logging Integration**: Each module now records important events (e.g., sign-in attempts, onboarding steps, transactions) into a dedicated log system.
3. **Data Persistence**: The system uses JSON files to store user data, credentials, and transactions; sensitive fields are **AES-encrypted**.
4. **Security Measures**: OTP verification, account blocking upon repeated failures, forced password changes after a certain login count.

---

2. Core Modules

2.1 Main Module

Purpose:

- Acts as the **entry point** for the application.
- Provides a **text-based menu** for **Sign In** (existing users) or **Sign Up** (new users).
- Orchestrates the flow after successful sign-in, directing the user to **Banking Operations** or **Banking Services**.

Key Responsibilities:

1. **Greetings**: Shows a welcome message to the user.
2. **Sign In / Sign Up Choice**:
    - If the user selects sign-up, it invokes **Customer_Onboarding**.
    - If sign-in, it calls the **SignIn** class.
3. **Post-Login Options**: A second menu prompting the user to choose Banking Operations or Banking Services.

4. **Logging Integration**:
    - Log each session start ("Application started"), sign-in attempt, sign-up attempt, and final exit message.

2.2 Customer_Onboarding

Purpose:

- Handles **registration** of new customers, ensuring they meet the **age** and **pincode** requirements.
- Validates email and mobile number using an **OTP** process.
- Creates an encrypted account password and initializes user data in JSON files.

Detailed Workflow:

1. **form()**:
    - Collects salutation, name, date of birth (must yield age 18–65), address, pincode (^4[0-4]\d{4} for Maharashtra), email (only Gmail), and mobile number.
    - If any validation fails (e.g., age out of range), the process aborts.
2. **Account Number Generation**:
    - Creates an 11-digit random number.
3. **Contact Updates**:
    - Optionally updates email/mobile before final OTP checks.
4. **OTP Verification** (Mobile & Email):
    - Each has **2 attempts**.
    - Failure triggers the system to delete the partially created account data.
5. **Default Password**:
    - Generates a 12-character complex password (uppercase, lowercase, digits, special chars).
6. **Data Persistence**:
    - Writes user info to **account_details.json**, credentials (pwd, count=0, account_blocked=None) to **credentials.json**, and initializes transactions.json with a zero-balance record.
7. **Welcome Email**:
    - Sends account number and password to the user's email.

Logging:

- **Onboarding Start**: “Customer onboarding initiated.”
- **Form Data**: “User details collected for [Name].”
- **OTP**: “Email/Mobile OTP sent to [email/mobile].” Log success or failure.
- **Account Creation**: “Account [account_number] created successfully” or “Onboarding failed, data deleted.”

2.3 SignIn

Purpose:

- Authenticates existing users and **manages account blocking** or forced password changes.
- Allows up to **3 attempts** before blocking for **8 hours**.
- Every **25 successful logins** triggers a **forced password change**.

Key Methods:

1. **log_in(account_number)**:
    - Prompts for password up to 3 times.
    - If the user fails thrice, the account is **blocked for 8 hours**.
2. **change_password(data)**:
    - When data['count'] % 25 == 0, user must provide the old password and set a new one.
    - Up to 3 tries; else blocked for 8 hours.

Logging:

- **Login Attempts**: “Sign-in attempt #N for account [account_number].”
- **Force Password Change**: “Forcing password update for account [account_number].”
- **Block Events**: “Account [account_number] blocked until [timestamp].”
- **Successful Login**: “Account [account_number] sign-in success.”

2.4 BankingOperations

Purpose:

- Implements **deposit**, **withdraw**, and **account statement** functionalities.

Key Functions:

1. **deposit(account_number)**:
    - Asks for deposit amount, increments balance, and logs a transaction entry with date/time.
2. **withdraw(account_number)**:
    - Checks current balance and transaction limit (if set).
    - Sends an OTP to the user's email.
        - If OTP fails after 2 tries, blocks the account for **2 hours**.
        - If success, deducts the amount and logs the transaction.
3. **account_statement(account_number)**:
    - Returns a list of the last 5 transactions by default, or can handle date-based queries in tandem with the **display_and_save()** method.
4. **display_and_save(account_number)**:
    - Optionally saves the statement to a .txt file (downloads folder).

Logging:

- **Deposit**: "Deposit of [amount] to account [account_number]. New balance: [balance]."
- **Withdrawal**: "Withdrawal request of [amount]. OTP success/fail. If fail, block for 2 hours."
- **Statements**: "Account [account_number] statement displayed/saved."

2.5 BankingSerivices

Purpose:

- Offers additional banking features like passbook downloads, profile editing, transaction limits, and nominee addition.

Functions:

1. **passbook_download(account_number)**:
    - Handles statements for **last 30, 60, 90, 365 days**, or a custom date.
2. **account_service(account_number)**:
    - User can set a **transaction limit** stored in credentials.json.
3. **edit_profile(account_number)**:
    - Modify contact details, address, etc., with OTP re-validation for security.
4. **add_nominee(account_number)**:

    - If balance ≥ 50,000, the user can add a nominee.

Logging:

- **Passbook**: “Passbook downloaded for account [account_number] covering [date_range].”
- **Profile Edit**: “Profile info updated for account [account_number]. OTP validated.”
- **Nominee**: “Nominee added for account [account_number].”

2.6 Logging Module

Purpose:

- A dedicated system that **records important events** across the entire application.
- Enhances maintainability and debugging by providing a **timeline** of user actions.

Implementation Approach:

- Use Python’s built-in **logging** library:

python

Copy code

```
import logging


logging.basicConfig(
    filename='bank_app.log',
    level=logging.INFO,
    format='%(asctime)s [%(levelname)s] %(message)s'
)
```

- Replace print() statements for critical events with logging.info(), logging.warning(), etc.

Typical Log Events:

- Sign-in attempts, success/failure
- Password changes (forced or user-initiated)
- OTP generation/validation success or failure
- Deposits, withdrawals
- Account blocks with timestamps
- Profile edits, passbook downloads

Benefits:

- Provides an **audit trail** for security, compliance, or debugging.
- Students can learn how professional software logs user actions and system states.

2.7 DataBase

Purpose:

- Handles JSON-based data storage.
- Reads & writes to the following JSON files:
    1. **account_details.json**: Personal info (e.g., names, addresses, pincode, email, mobile, nominee).
    2. **credentials.json**: Encrypted password, login count, account_blocked timestamp, transaction limit.
    3. **transactions.json**: A timestamped list of each deposit/withdrawal transaction.

Key Methods:

- **dump_data_into_database() / dump_accont_credentials_into_database()**: Initial creation of user or credential records.
- **update_user_details() / update_credentials() / update_transcations()**: Edits existing entries.
- **get_user_details(account_number, file_name)**: Fetches user info from the specified JSON.
- **delete_account_number(account_number)**: Removes a partially created account if onboarding fails.

Logging (Optional Examples):

- "Dumped account details for [account_number] to account_details.json."
- "Credentials updated for [account_number]."

2.8 Otp

Purpose:

- Generates and validates **mobile** or **email** OTPs.
- If the user fails OTP verification within 2 attempts, accounts may be **blocked** or **deleted** (in onboarding) based on context.

Mobile OTP:

- 6-digit numeric code sent via **Twilio**.
- If fail, account might get blocked for 48 hours (the code's message) or 2 hours if triggered in a **withdrawal** scenario.

Email OTP:

- 6-character mix of uppercase, lowercase, digits, and special characters.
- Sent via SMTP, also 2 attempts. Failure leads to block/deletion depending on scenario.

Logging:

- "Generated OTP for account [account_number] – mobile/email."
- "OTP success/failure on attempt #n."

2.9 Crypto_encryption

Purpose:

- AES encryption for sensitive fields (account number, password, email).
- Uses a 32-byte key stored in encryption_key.bin.
- Encryption mode: **AES-CBC** with a zeroed IV.

Methods:

- **encrypt_value(value)**: Pads plaintext, encrypts, returns base64 ciphertext.
- **decrypt_value(ciphertext_base64)**: Decrypts base64 ciphertext, strips padding, returns plaintext.

Logging:

- Typically minimal: "Encryption key retrieved," or "Data encrypted." Enough to track key usage but not store sensitive plaintext in logs.

2.10 Validator

Purpose:

- Ensures **strict input checks** to maintain data integrity (names, addresses, dates, email, mobile format, password complexity, amounts, etc.).

Key Validations:

1. **DOB** → Must place the user age at 18–65.
2. **Email** → Must match a pattern for **Gmail**.

3. **Pincode** → Regex enforcing a Maharashtra range (^4[0-4]\d{4}).
4. **Password** → At least 8 characters, containing uppercase, lowercase, digit, and special char from !@#%^&*.
5. **Amount** → Must be numeric and non-zero (and if withdrawal, must not exceed balance).

Logging (Optional):

- "Invalid input for DOB," or "Email validated successfully."
- This can help debug repeated invalid entries.

---

3. Logging: Cross-Cutting Implementation

Global Logging:

- A single **logging configuration** in a central file or an initialization block sets level, format, and log file name (bank_app.log).
- Each module **imports** logging and logs relevant events. This ensures a unified log style.

Recommended Logging Levels:

- logging.INFO for standard operation messages (sign-ins, deposits).
- logging.WARNING for suspicious events (OTP failure, user nearing block).
- logging.ERROR for account blocks, critical OTP failures, or password mismatch after multiple attempts.

Benefits:

- Students learn to trace user journeys from onboarding through daily banking usage.
- Logs provide a **timeline** of everything from code perspective.

---

4. Overall Behavior and Block Durations

1. **Onboarding OTP Fail** → The code **deletes** partial entries in account_details.json.
2. **Sign-In Fail (3 tries)** → Account blocked for **8 hours**.
3. **Withdrawal OTP Fail (2 tries)** → Account blocked for **2 hours**.
4. **Logging** each block event: "Account [X] blocked until [timestamp]."

These distinct durations illustrate varied security policies. Students can unify them or keep them separate to see how **requirements** sometimes differ in real-world applications.

---

5. Future Extensions

1. **Advanced Log Management**:
   - **Rotating logs** (via TimedRotatingFileHandler) if the log file grows large.
   - **Log analytics** to detect suspicious patterns (multiple OTP failures).
2. **Switching JSON to an SQL database**:
   - For scalability, or to teach relational DB schemas.
3. **Testing**:
   - Implement a suite of **unit tests** (using pytest or unittest) for the OTP, encryption, and deposit/withdraw logic.
4. **Web Interface**:
   - Could extend your console-based system to a **Flask** or **Django** web interface, still reusing the same modules.

Conclusion

With the **logging module** integrated, your **Bank of Python** project achieves a more production-like feel. Students will learn:

- **Modular Architecture**: Clear division of concerns across classes (Onboarding, SignIn, BankingOperations, etc.).
- **Data Validation & Encryption**: The interplay of Validator and Crypto_encryption ensures robust data handling.
- **OTP & Account Blocking**: Realistic security flows, reinforcing safe coding practices.
- **Logging**: Auditing and debugging become much easier, teaching them professional standards for **observability** in software.

Use this document to guide students. Emphasize how logging transforms a rudimentary console application into a more **auditable, traceable system**, closer to the standards of real-world banking software.