Test-Driven Development (TDD) approach for API testing and integration scenarios, focusing on various use cases for spends, investments, EPF balance, liabilities, etc.

---

1. Smoke Test Suite

Purpose: Verify critical paths, basic functionality, and API accessibility.

Use Cases:

- **Authentication API**:
    - Verify valid and invalid login credentials.
    - Test token generation and expiration.
- **Bank Spend API**:
    - Verify if the API is accessible and returns a 200 response.
    - Check for mandatory fields in requests, such as account number and date range.
- **Investment Portfolio API**:
    - Validate the response structure for mutual funds and stocks endpoints.
    - Ensure the API returns a summary of investments without calculations.
- **EPF and Liabilities API**:
    - Verify connectivity and basic response from the EPF provider.
    - Test loan balance retrieval API for a valid user ID.

---

2. Regression Test Suite

Purpose: Ensure that new changes don't break existing functionality.

Use Cases:

- **Authentication API**:
    - Validate multi-factor authentication with various configurations (e.g., OTP, biometrics).
    - Ensure session logout API removes access tokens.
- **Bank Spend API**:
    - Validate transaction categorization (e.g., dining, fuel) based on merchant codes.
    - Test for correct handling of date ranges (e.g., monthly, yearly).
    - Test pagination for large transaction histories.
- **Investment Portfolio API**:

    - Validate detailed breakdowns for mutual funds, including fund names, NAV, and quantity.
    - Check integration with third-party stock price providers for accuracy.
- **EPF and Liabilities API**:
    - Ensure accurate calculation of total EPF balance, including employer and employee contributions.
    - Test the integration with loan providers for fetching details of EMIs, due dates, and outstanding amounts.

---

3. Integration Test Suite

Purpose: Validate end-to-end workflows involving multiple systems.

Use Cases:

- **Spends Integration**:
    - Simulate a user authorizing the app to access their bank account.
    - Verify spends data aggregation across multiple accounts (e.g., credit cards and savings accounts).
    - Test for data consistency when transactions are updated at the bank's end.
- **Investments Integration**:
    - Validate integration with multiple investment providers (e.g., mutual fund registrars, stock exchanges).
    - Test scenarios where a user has investments in multiple asset classes (e.g., equity, debt).
    - Handle corner cases like investments sold or portfolios with zero holdings.
- **EPF Integration**:
    - Test the workflow of fetching EPF balance after the user inputs their UAN and password.
    - Simulate EPF portal downtime and validate retry logic or fallback mechanisms.
- **Liabilities Integration**:
    - Verify the integration of loan APIs for fetching data across personal, car, and home loans.
    - Validate workflows for updating outstanding balances after partial payments.

---

4. Functional Test Suite

Purpose: Validate that each API feature works as expected.

Use Cases:

- **Authentication**:
    - Test API rate limiting for login attempts.
    - Validate token refresh workflows.
- **Bank Spend**:
    - Validate currency conversion for spends in different currencies.
    - Check filtering options like "spends by category" or "by date."
- **Investments**:
    - Ensure performance metrics like CAGR or annual returns are accurate.
    - Test invalid fund codes and missing stock tickers.
- **EPF**:
    - Check partial withdrawal scenarios.
    - Validate cases where UAN is inactive or incorrect.
- **Liabilities**:
    - Verify scenarios with multiple loans under the same user ID.
    - Handle cases where loans are closed but remain listed.

---

5. Negative Test Suite

Purpose: Test error handling, boundary cases, and resilience.

Use Cases:

- **Authentication API**:
    - Test invalid tokens, expired tokens, and unauthorized access.
    - Handle API responses for blacklisted accounts.
- **Bank Spend API**:
    - Simulate invalid account numbers and unsupported bank integrations.
    - Test extreme date ranges or malformed requests.
- **Investment API**:
    - Verify API responses when no data is returned (e.g., invalid user).
    - Handle incorrect ISIN or mutual fund scheme IDs.
- **EPF API**:

    - Simulate incorrect UAN and invalid credentials.
    - Test scenarios where the EPF portal is unreachable.
- **Liabilities API**:
    - Handle non-existent loan IDs or incorrect borrower credentials.
    - Simulate API timeouts from third-party providers.

---

6. Performance Test Suite

Purpose: Test for API responsiveness, scalability, and throughput.

Use Cases:

- Validate response time for fetching spends data for a high-volume account (e.g., 1,000 transactions/month).
- Test API performance under concurrent user loads accessing investment portfolios.
- Benchmark EPF balance retrieval under varying server loads.
- Ensure APIs handling liabilities can process a high number of EMI calculations simultaneously.

TDD Approach Implementation

1. **Define the Requirements**: Collaborate with stakeholders to finalize the expected behavior of each API.
2. **Write Tests First**: Create unit tests for each API functionality before writing the implementation code.
3. **Iterative Development**:
    - Write just enough code to pass the test.
    - Refactor for performance and scalability.
    - Repeat for each new functionality.
4. **Automate Testing**:
    - Use frameworks like RestAssured (Java) or Postman for API tests.
    - Integrate tests into CI/CD pipelines with Jenkins or Azure DevOps.

Expanded API Contracts for Integrations

1. Bank Account Spending Data Integration

- **Endpoint**: GET /bank/spending
- **Headers**:

- Authorization: Bearer {access_token}

- **Query Parameters**:
  - account_id: The unique identifier for the user's bank account.

- **Response**:


```
{
  "total_balance": 5000.00,
  "transactions": [
    {"date": "2024-12-01", "amount": 50.00, "merchant": "SuperMart", "category": "Groceries"},
    {"date": "2024-12-02", "amount": 200.00, "merchant": "FlightCo", "category": "Travel"}
  ],
  "available_balance": 4800.00
}
```


2. Investment Data Integration

- **Endpoint**: GET /investments
- **Headers**:
  - Authorization: Bearer {access_token}
- **Query Parameters**:
  - investment_type: (optional) Filter by investment type (e.g., "stocks", "mutual funds", "bonds")
- **Response**:


```
{
  "total_investments": 25000.00,
  "investment_summary": [
    {"type": "Mutual Fund", "amount": 15000.00, "current_value": 16000.00},
    {"type": "Stocks", "amount": 5000.00, "current_value": 4800.00},
    {"type": "Bonds", "amount": 5000.00, "current_value": 5000.00}
  ],
  "top_performing_investment": {"type": "Mutual Fund", "growth": 6.67}
}
```


3. EPF (Employee Provident Fund) Balance

- **Endpoint**: GET /epf/balance

- **Headers**:
    - Authorization: Bearer {access_token}
- **Response**:


```
{
  "epf_balance": 12000.00,
  "total_contributions": 10000.00,
  "interest_earned": 2000.00
}
```


4. Liabilities Data Integration (Loans, etc.)

- **Endpoint**: GET /liabilities
- **Headers**:
    - Authorization: Bearer {access_token}
- **Response**:


```
{
  "total_liabilities": 15000.00,
  "liability_details": [
    {"type": "Home Loan", "remaining_balance": 10000.00, "monthly_payment": 1200.00, "interest_rate": 4.5},
    {"type": "Car Loan", "remaining_balance": 5000.00, "monthly_payment": 500.00, "interest_rate": 5.0}
  ]
}
```


Use Case for Integrations

Use Case: A user wants a complete view of their financial health, including spending habits, investments, EPF balance, and liabilities.

Scenario:

1. **Client Application**: The financial management app needs to aggregate data from different sources to give users a clear picture of their finances.
2. **Integration Workflow**:
    - **Bank Integration**: The app calls the /bank/spending endpoint to get recent spending data and current account balance.
    - **Investment Integration**: The app queries the /investments endpoint to retrieve investment details and their current values.

    - **EPF Integration**: The app accesses the /epf/balance endpoint to check the EPF balance and interest earned.
    - **Liabilities Integration**: The app uses the /liabilities endpoint to understand outstanding debts, payment obligations, and interest rates.
3. **Response Handling**:
    - The app combines the data from each endpoint to create a dashboard that provides the user with an overview of total assets, liabilities, and net worth.
4. **User Insights**:
    - The app may include visualizations such as charts showing spending trends, investment growth, liabilities breakdown, and net worth over time.

Security Considerations

- **OAuth 2.0 and Token Expiry**: Ensure secure authentication and periodically refresh tokens to maintain access.
- **Data Encryption**: Use HTTPS for data transmission and encrypt sensitive data in storage.
- **Rate Limiting**: Implement rate limiting to prevent abuse and ensure fair use.