Bank Adapter Interface / Base Module

Goal: Create a modular system to connect to multiple banks and mobile money providers.

Checklist:

- Create folder: backend/src/universal-payment-platform/Adapters/
- Create interface IBankAdapter.cs with methods:
    - Task<PaymentResponse> InitiatePayment(PaymentRequest request)
    - Task<PaymentStatus> CheckStatus(string transactionId)
    - Task<RefundResponse> Refund(string transactionId)
- Create **mock adapter** MockBankAdapter.cs that implements IBankAdapter
- Ensure mock adapter returns **fake success/failure responses**
- Unit test the mock adapter

---

2. Payment Orchestration Engine

Goal: Central module to handle all payment requests and route them to the correct adapter.

Checklist:

- Create folder: backend/src/universal-payment-platform/Services/
- Create class PaymentOrchestrator.cs
- Add method: Task<PaymentResponse> ProcessPayment(PaymentRequest request)
- Inject adapters into orchestrator (DI via constructor)
- Route payment requests to the correct adapter based on PaymentType (Bank or Mobile Money)
- Add retry logic for transient failures
- Add centralized logging for every payment request
- Unit test orchestrator with **mock adapters**

3. Logging & Error Handling

Goal: Ensure all operations are logged and failures are captured.

Checklist:

- Install and configure logging library (Serilog recommended)
- Create folder: backend/src/universal-payment-platform/Infrastructure/Logging/
- Add structured logging for:
    - Payment initiation
    - Payment success/failure
    - Refunds
    - Adapter exceptions
- Centralized exception handling middleware (ExceptionMiddleware.cs)
- Ensure **all unhandled exceptions** return standardized error responses

---

4. Data Models & Database

Goal: Define the structure of data for payments and transactions.

Checklist:

- Create folder: backend/src/universal-payment-platform/Models/
- Define models:
    - PaymentRequest.cs (amount, payer, recipient, payment type)
    - PaymentResponse.cs (transaction ID, status, message)
    - PaymentStatus.cs (Pending, Completed, Failed)
    - RefundResponse.cs (transaction ID, status, message)
    - TransactionLog.cs (ID, type, status, timestamps, details)
- Create DB context: PaymentDbContext.cs
- Configure EF Core / PostgreSQL / SQL Server
- Create migrations and initialize database
- Unit test CRUD operations

Optional Extra: API Layer

- Create Controllers/PaymentController.cs
- Expose endpoints:
    - POST /api/payments → calls PaymentOrchestrator.ProcessPayment
    - GET /api/payments/{id}/status → calls PaymentOrchestrator.CheckStatus
    - POST /api/payments/{id}/refund → calls PaymentOrchestrator.Refund
- Test endpoints with **Postman or Swagger**

Frontend Setup

Goal: Initialize Angular project and prepare dev environment.

Checklist:

- Navigate to frontend folder:
- cd C:\Users\cliff\Desktop\platform\frontend
- Create Angular project (if not done yet):
- ng new universal-payment-ui
    - Select **routing**: Yes
    - Stylesheet format: SCSS (optional)
- Open project in VS Code:
- code .
- Install dependencies:
- npm install
- Run dev server:
- ng serve
- Verify http://localhost:4200 loads the default Angular page

---

1. Core Modules & Services

Goal: Connect frontend to backend via services and provide reusable components.

Checklist:

- Create folder: src/app/services/
- Create payment.service.ts
    - Methods: initiatePayment(), checkStatus(), refundPayment()
    - Use Angular **HttpClient** to call backend endpoints
- Create folder: src/app/models/
    - Models: PaymentRequest, PaymentResponse, PaymentStatus
- Set up environment variables for API base URL (src/environments/environment.ts)

---

2. Payment Dashboard Components

Goal: Build UI for creating and monitoring payments.

Checklist:

- Create folder: src/app/components/
- Component: payment-form
    - Fields: amount, payer, recipient, payment type (bank/mobile money)
    - Submit button calls payment.service.initiatePayment()
- Component: payment-status
    - Displays status of a transaction
    - Calls payment.service.checkStatus()
- Component: transaction-log
    - Table of recent transactions
    - Display: transaction ID, type, amount, status, timestamp

---

3. Navigation & Layout

Goal: Provide a clean UI structure for users.

Checklist:

- Create app-routing.module.ts routes:

    - /payments → payment form
    - /status → payment status
    - /logs → transaction logs
- Create navbar component with links to pages
- Optionally use Angular Material for styling and responsive design

---

4. Integration & Testing

Goal: Connect frontend to backend and verify functionality.

Checklist:

- Ensure backend server is running (dotnet run)
- Test payment form → submits to orchestrator (mock or real)
- Test status check → updates UI correctly
- Test refund → status and logs update as expected
- Handle errors gracefully (display messages)

---

5. Optional Enhancements

- Add **loading spinners** during API calls
- Add **notifications** for success/failure (Angular Material Snackbar)
- Add **role-based access** (admin vs user)
- Add **charts for transaction analytics** (ng2-charts, Chart.js)
