🚀Banking App – UI & Microservices Integration Routine (Step-by-Step)

This guide takes you from a fresh **TailAdmin React** UI to a fully integrated frontend talking to your **Spring Boot microservices** via **API Gateway**. It assumes you already have these repos:

- **user-service**, **account-service**, **transaction-service**, **notification-service**
- **api-gateway** (Spring Cloud Gateway), **discovery-service** (Eureka)
- **banking-ui** (React + Tailwind via TailAdmin)

> If you're new to React/Tailwind: each step includes minimal "what/why/how" so you can implement it yourself.

---

0) Prerequisites & Ports

- **Node.js** LTS installed (check: `node -v`, `npm -v`).
- **Ports (suggested)**
- discovery-service: **8761**
- api-gateway: **8080**
- user-service: **9001**
- account-service: **9002**
- transaction-service: **9003**
- notification-service: **9004**
- UI (React dev server): **3000**

> UI calls **only the gateway** (http://localhost:8080). Services talk among themselves through Eureka.

---

1) Run TailAdmin React UI

1. In `banking-ui/`:

```
npm install
npm start
```

2. Confirm it runs at http://localhost:3000.
3. Quick Tailwind primer:
4. Utility classes like `p-4`, `bg-gray-100`, `rounded-xl` are added **directly to HTML/JSX** elements.
5. You rarely write custom CSS; you compose styles with classes.

---

2) Clean Up & Project Structure

Minimal structure to build features cleanly:

```
src/
  api/                    # Axios instance & service modules
    apiClient.js
    authApi.js
    accountApi.js
    transactionApi.js
    notificationApi.js
  components/             # Reusable UI parts
    Layout/
      Sidebar.jsx
      Topbar.jsx
      ProtectedRoute.jsx
      RoleRoute.jsx
  context/
    AuthContext.jsx
  pages/
    Login.jsx
    Register.jsx
    Dashboard.jsx
    Accounts.jsx
    Transactions.jsx
    Profile.jsx
    Admin/
      Users.jsx
      TransactionsAdmin.jsx
  utils/
    jwt.js                # helpers (decode, isExpired)
  App.jsx
  main.jsx (or index.jsx)
```

> Keep TailAdmin’s layout/shell; remove demo pages you don’t need.

---

3) Environment Config (.env)

Create `.env` in project root (and `.env.example` for the repo):

```
VITE_API_BASE_URL=http://localhost:8080
# if CRA instead of Vite, use: REACT_APP_API_BASE_URL=
```

Use it in code via `import.meta.env.VITE_API_BASE_URL` (Vite) or `process.env.REACT_APP_API_BASE_URL` (CRA).

---

4) API Layer (Axios + Interceptors)

`src/api/apiClient.js`

```
import axios from 'axios';

const api = axios.create({
  baseURL: import.meta.env.VITE_API_BASE_URL, // or
process.env.REACT_APP_API_BASE_URL
  timeout: 15000,
});

// Attach JWT from localStorage
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});

// Global 401 handler → logout/redirect to login
api.interceptors.response.use(
  (res) => res,
  (err) => {
    if (err?.response?.status === 401) {
      localStorage.removeItem('token');
      window.location.href = '/login';
    }
    return Promise.reject(err);
  }
);

export default api;
```

Service modules (example: `src/api/authApi.js`)

```
import api from './apiClient';

export const login = (username, password) =>
  api.post('/users/login', { username, password });
```

```
export const register = (payload) =>
  api.post('/users/register', payload);

export const me = () => api.get('/users/me'); // optional: current user endpoint
```

Other modules:

```
// accountApi.js
import api from './apiClient';
export const getMyAccounts = () => api.get('/accounts/me');
export const getAccountById = (id) => api.get(`/accounts/${id}`);
export const createAccount = (payload) => api.post('/accounts', payload);

// transactionApi.js
import api from './apiClient';
export const listTransactions = (params) => api.get('/transactions', {
params });
export const deposit = (payload) => api.post('/transactions/deposit', payload);
export const withdraw = (payload) => api.post('/transactions/withdraw',
payload);
export const transfer = (payload) => api.post('/transactions/transfer',
payload);

// notificationApi.js
import api from './apiClient';
export const sendEmail = (payload) => api.post('/notifications/email', payload);
```

> **Note**: All paths are **gateway routes** (e.g., `/users/**`, `/accounts/**`). You'll configure these routes in Spring Cloud Gateway.

---

5) Auth Context, Protected Routes & Roles

`src/context/AuthContext.jsx`

```
import { createContext, useContext, useEffect, useState } from 'react';
import { login as loginApi } from '../api/authApi';
import { decodeJwt, isExpired } from '../utils/jwt';

const AuthContext = createContext(null);
export const useAuth = () => useContext(AuthContext);

export default function AuthProvider({ children }) {
  const [user, setUser] = useState(null); // { username, roles }
  const [token, setToken] = useState(localStorage.getItem('token'));

  useEffect(() => {
    if (!token) return setUser(null);
    const payload = decodeJwt(token);
    if (!payload || isExpired(payload)) {
      localStorage.removeItem('token');
      setToken(null);
      setUser(null);
    } else {
      setUser({ username: payload.sub, roles: payload.roles ||
payload.authorities || [] });
    }
  }, [token]);

  const login = async (username, password) => {
    const { data } = await loginApi(username, password);
    // backend should return { token: '...' }
    localStorage.setItem('token', data.token);
    setToken(data.token);
  };

  const logout = () => {
    localStorage.removeItem('token');
    setToken(null);
    setUser(null);
  };

  const hasRole = (role) => user?.roles?.includes(role);

  return (
    <AuthContext.Provider value={{ user, token, login, logout, hasRole }}>
      {children}
    </AuthContext.Provider>
  );
}
```

JWT helpers – `src/utils/jwt.js`

```
export const decodeJwt = (token) => {
  try {
    const payload = token.split('.')[1];
    return JSON.parse(atob(payload));
  } catch {
    return null;
  }
};

export const isExpired = (payload) => {
  if (!payload?.exp) return true;
  const nowSec = Math.floor(Date.now() / 1000);
  return payload.exp < nowSec;
};
```

Protected routes – `components/Layout/ProtectedRoute.jsx`

```
import { Navigate } from 'react-router-dom';
import { useAuth } from '../../context/AuthContext';

export default function ProtectedRoute({ children }) {
  const { token } = useAuth();
  if (!token) return <Navigate to="/login" replace />;
  return children;
}
```

Role guard – `components/Layout/RoleRoute.jsx`

```
import { Navigate } from 'react-router-dom';
import { useAuth } from '../../context/AuthContext';

export default function RoleRoute({ children, role }) {
  const { user } = useAuth();
  if (!user?.roles?.includes(role)) return <Navigate to="/" replace />;
  return children;
}
```

---

6) Routing (React Router)

Basic router in `App.jsx`:

```
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import AuthProvider from './context/AuthContext';
import ProtectedRoute from './components/Layout/ProtectedRoute';
import RoleRoute from './components/Layout/RoleRoute';
import Login from './pages/Login';
import Register from './pages/Register';
import Dashboard from './pages/Dashboard';
import Accounts from './pages/Accounts';
import Transactions from './pages/Transactions';
import Users from './pages/Admin/Users';

export default function App() {
  return (
    <AuthProvider>
      <BrowserRouter>
        <Routes>
          <Route path="/login" element={<Login />} />
          <Route path="/register" element={<Register />} />

          <Route path="/" element={<ProtectedRoute><Dashboard /></
ProtectedRoute>} />
          <Route path="/accounts" element={<ProtectedRoute><Accounts /></
ProtectedRoute>} />
          <Route path="/transactions" element={<ProtectedRoute><Transactions /
></ProtectedRoute>} />

          <Route path="/admin/users" element={
            <ProtectedRoute>
              <RoleRoute role="ADMIN"><Users /></RoleRoute>
            </ProtectedRoute>
          } />
        </Routes>
      </BrowserRouter>
    </AuthProvider>
  );
}
```

---

7) Pages (Skeletons you can expand)

Login.jsx

```
import { useState } from 'react';
import { useAuth } from '../context/AuthContext';

export default function Login() {
  const { login } = useAuth();
  const [form, setForm] = useState({ username: '', password: '' });
  const [error, setError] = useState('');

  const onSubmit = async (e) => {
    e.preventDefault();
    try {
      await login(form.username, form.password);
      window.location.href = '/';
    } catch (err) {
      setError('Invalid credentials');
    }
  };

  return (
    <div className="min-h-screen flex items-center justify-center bg-gray-50">
      <form onSubmit={onSubmit} className="bg-white p-8 rounded-2xl shadow w-
full max-w-md space-y-4">
        <h1 className="text-2xl font-bold">Sign in</h1>
        {error && <div className="text-red-600 text-sm">{error}</div>}
        <input className="w-full border p-3 rounded-xl" placeholder="Username"
value={form.username}
          onChange={(e) => setForm({ ...form, username: e.target.value })} />
        <input type="password" className="w-full border p-3 rounded-xl"
placeholder="Password" value={form.password}
          onChange={(e) => setForm({ ...form, password: e.target.value })} />
        <button className="w-full p-3 rounded-xl bg-black text-white">Login</
button>
      </form>
    </div>
  );
}
```

Transactions.jsx (list with pagination & filters)

```
import { useEffect, useState } from 'react';
import { listTransactions } from '../api/transactionApi';

export default function Transactions() {
  const [rows, setRows] = useState([]);
  const [page, setPage] = useState(0);
  const [size, setSize] = useState(10);
  const [filters, setFilters] = useState({ type: '', from: '', to: '' });

  const load = async () => {
    const { data } = await listTransactions({ page, size, ...filters });
    setRows(data.content || data); // support pageable or raw
  };

  useEffect(() => { load(); }, [page, size]);
  const onFilter = (e) => { e.preventDefault(); setPage(0); load(); };

  return (
    <div className="p-6 space-y-4">
      <form onSubmit={onFilter} className="flex gap-4 items-end">
        <select className="border p-2 rounded-xl" value={filters.type}
onChange={(e)=>setFilters(f=>({...f,type:e.target.value}))}>
          <option value="">All</option>
          <option value="DEPOSIT">Deposit</option>
          <option value="WITHDRAW">Withdraw</option>
          <option value="TRANSFER">Transfer</option>
        </select>
        <input type="date" className="border p-2 rounded-xl"
value={filters.from}
onChange={(e)=>setFilters(f=>({...f,from:e.target.value}))}/>
        <input type="date" className="border p-2 rounded-xl" value={filters.to}
onChange={(e)=>setFilters(f=>({...f,to:e.target.value}))}/>
        <button className="px-4 py-2 rounded-xl bg-black text-white">Apply</
button>
      </form>

      <div className="bg-white rounded-2xl shadow">
        <table className="w-full">
          <thead>
            <tr className="text-left">
              <th className="p-3">Date</th>
              <th className="p-3">Type</th>
              <th className="p-3">Amount</th>
              <th className="p-3">Account</th>
            </tr>
          </thead>
          <tbody>
            {rows.map(r => (
              <tr key={r.id} className="border-t">
                <td className="p-3">{new Date(r.timestamp).toLocaleString()}</
td>
                <td className="p-3">{r.type}</td>
                <td className="p-3">{r.amount}</td>
                <td className="p-3">{r.accountId}</td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>

      <div className="flex items-center gap-2">
        <button disabled={page===0} onClick={()=>setPage(p=>p-1)}
className="px-3 py-1 border rounded-xl">Prev</button>
```

```
        <span>Page {page+1}</span>
        <button onClick={()=>setPage(p=>p+1)} className="px-3 py-1 border
rounded-xl">Next</button>
        <select value={size} onChange={(e)=>setSize(+e.target.value)}
className="border p-1 rounded-xl">
          <option>5</option><option>10</option><option>20</option>
        </select>
      </div>
    </div>
  );
}
```

---

8) Gateway Routes & CORS (Backend)

`api-gateway/src/main/resources/application.yml` (example)

```
server:
  port: 8080

spring:
  application:
    name: api-gateway
  cloud:
    gateway:
      default-filters:
        - RemoveRequestHeader=Cookie
      globalcors:
        corsConfigurations:
          '[/**]':
            allowedOrigins: "http://localhost:3000"
            allowedMethods: "GET,POST,PUT,DELETE,OPTIONS"
            allowedHeaders: "*"
            allowCredentials: true
      routes:
        - id: user-service
          uri: lb://user-service
          predicates:
            - Path=/users/**
        - id: account-service
          uri: lb://account-service
          predicates:
            - Path=/accounts/**
        - id: transaction-service
          uri: lb://transaction-service
          predicates:
            - Path=/transactions/**
        - id: notification-service
          uri: lb://notification-service
          predicates:
            - Path=/notifications/**

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

- Each microservice registers with Eureka using `spring.application.name` matching the route URIs above.
- Secure paths in each service with Spring Security; permit `/users/register`, `/users/login`.

---

9) Microservices Security (Quick Recap)

- **user-service**: issues JWT on `/users/login`.
- Other services: validate JWT via a **JwtAuthenticationFilter**.
- Gateway: optionally validate JWT as a first line (advanced); simplest is to pass the `Authorization` header downstream.
- Ensure services have CORS enabled if you call them directly (we won't; UI → Gateway only).

---

10) Connect UI to APIs (End-to-End)

1. **Register → Login** (user-service via gateway)
2. UI `POST /users/register` → create user.
3. UI `POST /users/login` → receive `{ token }` → store in `localStorage`.
4. **Accounts** (account-service via gateway)
5. UI `GET /accounts/me` → show list.
6. UI `POST /accounts` → create new account.
7. **Transactions** (transaction-service via gateway)
8. UI `GET /transactions?page=0&size=10&type=DEPOSIT&from=2025-08-01&to=2025-08-31`.
9. UI `POST /transactions/transfer`.
10. **Notifications** (notification-service)
11. UI `POST /notifications/email` after success (or backend triggers on event).

    All requests carry `Authorization: Bearer <token>` via Axios interceptor.

---

11) Filters, Pagination, Export, Upload

- **Filters & Pagination**: Already shown in `Transactions.jsx`. Backend should accept query params (`page`, `size`, `sort`, `type`, `from`, `to`).
- **Export to Excel (UI)**: Quick client-side option using SheetJS:

```
npm install xlsx file-saver
```

```
import * as XLSX from 'xlsx';
import { saveAs } from 'file-saver';

const exportToExcel = (rows) => {
  const ws = XLSX.utils.json_to_sheet(rows);
  const wb = XLSX.utils.book_new();
  XLSX.utils.book_append_sheet(wb, ws, 'Transactions');
  const buf = XLSX.write(wb, { type: 'array', bookType: 'xlsx' });
  const blob = new Blob([buf]);
  saveAs(blob, 'transactions.xlsx');
};
```

- **File Upload (UI → Backend)**:

```
const upload = async (file) => {
  const form = new FormData();
  form.append('file', file);
  await api.post('/accounts/upload-statement', form, { headers: { 'Content-Type': 'multipart/form-data' } });
};
```

Backend endpoint parses `MultipartFile`.

---

12) Error, Loading, Toasts

- Add loading spinners (Tailwind: `animate-spin`) and error banners.
- Optional: `react-hot-toast` for toasts.

```
npm install react-hot-toast
```

---

13) GitHub Actions (UI)

`.github/workflows/ui-ci.yml`

```
name: UI CI
on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 20
      - run: npm ci
      - run: npm run build
```

> Later: add deploy (Netlify/Vercel/S3 + CloudFront).

---

14) Local Dev Routine (Daily)

1. Start **discovery-service** (Eureka).
2. Start **api-gateway**.
3. Start **user-service** → verify `/users/register`, `/users/login`.
4. Start **account-service**, **transaction-service**, **notification-service`**.
5. Run **banking-ui** (React) on port 3000.
6. Test flows in this order: register → login → accounts → transactions → notifications.

---

15) Security Notes

- Store JWT in `localStorage` for simplicity (acceptable for demos). For production, consider **HTTP-only cookies** + CSRF strategy at the gateway.
- Implement **account lock** after 3 failed logins (user-service), surface message to UI.
- Enforce **RBAC** in backend (method-level `@PreAuthorize`) and reflect in UI with `RoleRoute`.

---

16) Stretch Goals (Later)

- **React Query** for caching, loading states, retries.
- **Refresh tokens** (silent re-auth without forcing login).
- **Feature flags** (toggle modules).
- **Audit logs** page (admin-only), downloadable via Excel export.
- **Async notifications** via RabbitMQ/Kafka.

---

17) Checklist (Printable)

- [ ] TailAdmin runs locally
- [ ] .env configured with gateway URL
- [ ] Axios instance + interceptors
- [ ] AuthContext with login/logout, role support
- [ ] ProtectedRoute & RoleRoute wired
- [ ] Pages: Login, Register, Dashboard, Accounts, Transactions, Admin/Users
- [ ] Gateway routes + CORS
- [ ] Services discoverable in Eureka
- [ ] End-to-end: Register → Login → Accounts → Transactions
- [ ] Filters & pagination working
- [ ] Excel export working
- [ ] File upload working
- [ ] CI builds UI

Quick FAQ

- **Why gateway only?** Simpler CORS and security; one URL for UI.
- **Where do roles come from?** Encode roles in JWT `roles`/`authorities` claim when issuing token.
- **How to test without UI?** Use Postman against the gateway; then plug UI.

You're set! Follow the order above and you'll integrate every piece cleanly. If you get stuck on a step, paste the error/behavior and we'll debug fast.