# 1. Overall System Architecture

#### Frontend:

- **React Native (Expo)** app for user interaction, card display, and controls.
- Secure communication via HTTPS (TLS 1.2+).
- Token-based authentication (JWT or OAuth2)

#### Backend:

• **Node.js + Express** — main API handling business logic.

Structured modularly:

```
src/
routes/
controllers/
services/
models/
middlewares/
```

#### Infrastructure:

- Cloud hosting (AWS/GCP/Azure/Vercel/Render).
- PostgreSQL for persistent data.
- Redis for caching.
- Queue (RabbitMQ / Kafka) for async jobs.
- Vault or HSM for sensitive data.

# 2. Component-by-Component Implementation

# A. Mobile App (Frontend – Expo)

- Build UI for:
  - User signup/login (with OTP/email verification).
  - KYC submission (photo + ID upload).
  - Virtual card view (masked digits, expiry, CVV toggle).
  - o Card controls: enable/disable, set spending limits.
  - Transaction history.

Communicate with backend using Axios or Fetch:

```
const res = await fetch(`${API_URL}/cards/create`, {
  method: 'POST',
  headers: { Authorization: `Bearer ${token}` },
  body: JSON.stringify({ cardType: 'virtual' })
});
```

### **B. API Gateway / WAF**

- Use **NGINX** or **Cloudflare** in front of Express server for:
  - o Rate limiting
  - DDoS protection
  - Request filtering

Example NGINX config:

```
limit_req_zone $binary_remote_addr zone=mylimit:10m rate=5r/s;
```

# C. Backend (Auth + API)

- Express routes:
  - /auth/signup /auth/login
  - o /kyc/submit
  - /cards/create, /cards/freeze, /cards/transactions
- Use **Passport.js** or **jsonwebtoken** for auth.

```
Structure:
```

```
app.use('/auth', require('./routes/auth'));
app.use('/cards', require('./routes/cards'));
```

### **D. Card Provisioning Service**

Create a **Service Layer** (/services/cardProvisioning.js) that connects to third-party APIs like Stripe Issuing or Marqeta.

```
Example (Stripe Issuing):
```

return card;

```
import Stripe from 'stripe';

const stripe = new Stripe(process.env.STRIPE_SECRET);

export async function createVirtualCard(cardholderId) {

  const card = await stripe.issuing.cards.create({

    cardholder: cardholderId,

    type: 'virtual',

    currency: 'usd'

});
```

• Abstract it so that you can later plug in Marqeta or Visa APIs instead.

## E. Vault / Token Service

Use:

- HashiCorp Vault or AWS Secrets Manager.
- Store sensitive tokens (e.g., Stripe secret keys, PAN tokens).

Integration example:

```
import vault from 'node-vault';
const client = vault({ endpoint: process.env.VAULT_URL });
await client.write('secret/cards/stripe', { token: STRIPE_SECRET });
```

## F. KYC / AML Service

Options:

• Integrate 3rd-party services: **Sumsub**, **Trulioo**, **ShuftiPro**.

Backend endpoint:

```
status VARCHAR(20),
verified_at TIMESTAMP
);
```

### **G. Controls & Rules Engine**

• Implement spending controls, fraud detection, etc.

Node microservice or internal module:

```
if (transaction.amount > user.limit) return reject('Limit exceeded');
if (transaction.mcc === 'GAMBLING') return reject('Blocked category');
```

• Store configs in DB (controls table).

### H. Payment Reconciliation & Ledger

- Create scheduled jobs (via node-cron) to fetch transactions from card issuer and reconcile.
- Store in transactions table.

#### Example:

```
cron.schedule('*/30 * * * *', async () => {
  const txs = await stripe.issuing.transactions.list();
  await saveTransactionsToDB(txs);
});
```

#### I. Admin Dashboard

- Use a simple **React.js web app** (or Expo Web) connected to the same API.
- View KYC approvals, card lists, logs, and audit trails.

## J. Ops / Monitoring / Logging

- Winston or Pino for structured logs.
- **Prometheus + Grafana** for monitoring.

### Example:

```
logger.info('Card created successfully', { cardId, userId });
```

# K. Audit / Compliance

- Create immutable audit tables.
- Append-only DB logs (e.g., PostgreSQL with triggers).
- Store:
  - user actions
  - API responses
  - KYC decisions

### Example:

```
create table audit_log (

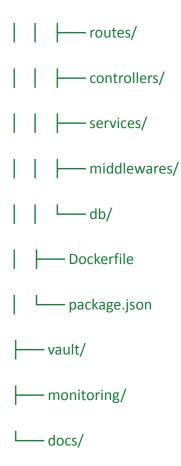
id Serial Primary Key,

user_id UUID,

action VARCHAR(255),

timestamp TIMESTAMP DEFAULT NOW()
);
```

# 3. Folder & Deployment Structure



### Deploy using:

- Docker + Kubernetes
- Use environment variables & secrets injection (via Vault or KMS)
- **CI/CD**: GitHub Actions for testing + deployment.

# 4. Flow Example (End-to-End)

- 1. User opens Expo app  $\rightarrow$  logs in.
- 2. App sends request to /cards/create.
- 3. Express backend calls the Card Provisioning Service (e.g., Stripe API).
- 4. Card data is tokenized & stored in Vault.
- 5. Backend sends masked card data to frontend.
- 6. User controls spending from app  $\rightarrow$  backend updates rules.

7. Reconciliation runs periodically via cron job.