Week 17.1

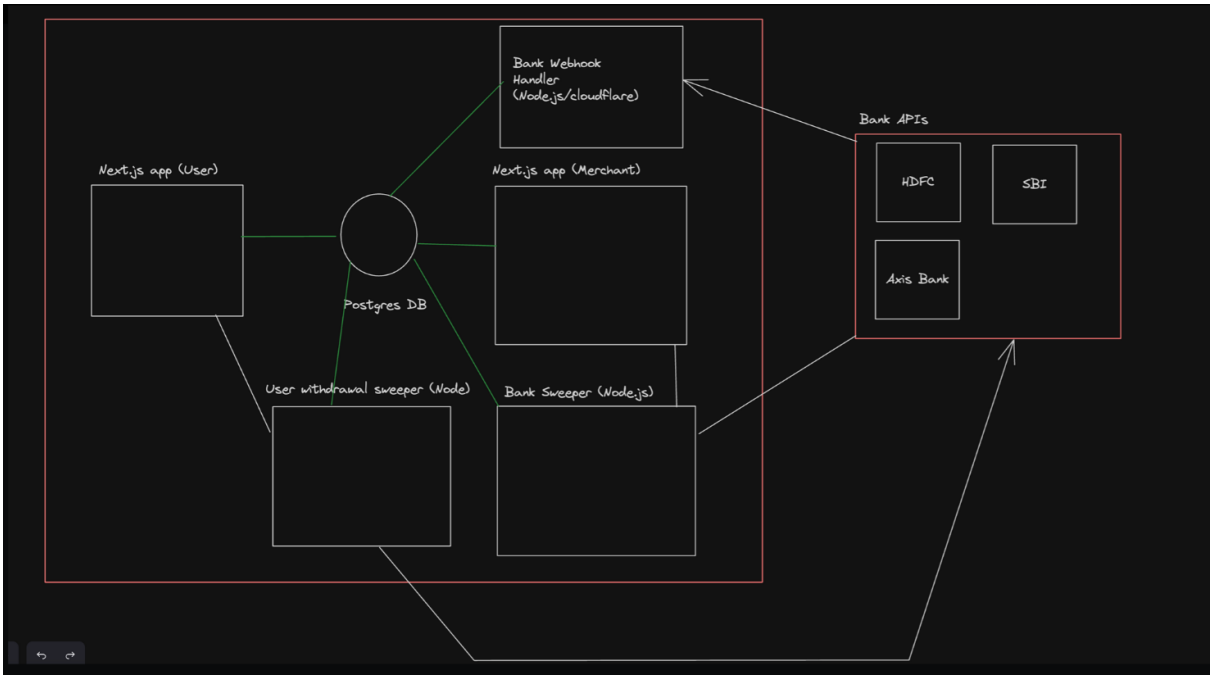
Building Paytm (1/3)

Up until now, our discussions have primarily revolved around theoretical concepts. In this lecture, Harkirat takes a `practical approach` by guiding us through the hands-on process of `building a Paytm like application`

The stack for this project includes Next.js for the frontend and backend (or a separate backend), Express for auxiliary backends, Turborepo for managing the monorepo, a PostgreSQL database, Prisma as the ORM, and Tailwind for styling.

> While there are `no specific notes` provided for this section, a mini guide is outlined below to assist you in navigating through the process of building the application. Therefore, it is strongly `advised to actively follow along` during the lecture for a hands-on learning experience.

Building Paytm (1/3)



Introduction

In this hands-on lecture, we will be building a Paytm-like application. The first step is to plan the features and design the user interface (UI) and user experience (UX). For the UX, we can either follow first principles or take inspiration from existing successful websites. As for the UI, while there are tools available, finding a good one can be challenging.

Next, we will focus on the high-level design, including the authentication provider, database, backend stack, frontend stack, and the modules we'll have (such as common, UI, and backend). We'll also decide on the cloud platform for deployment. Additionally, we'll delve into the low-level design, including schema design, route signatures, and frontend components. While entity-relationship diagrams can be useful, they may not be necessary unless you're a highly visual person.

Feature planning

User login

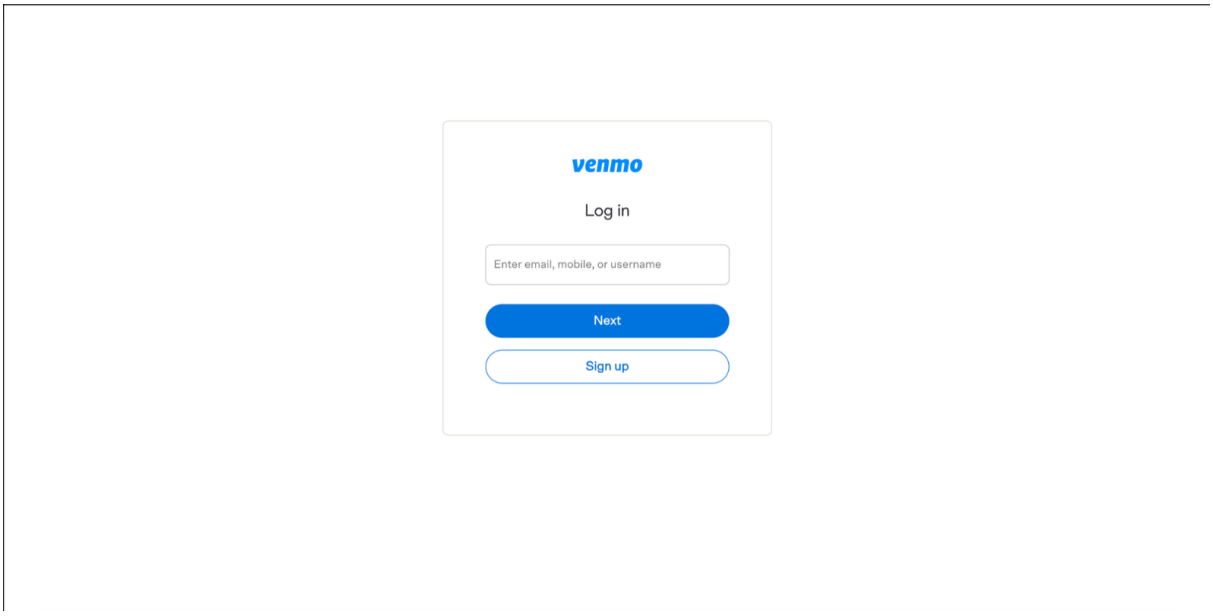
1. Auth (In this case, probably email/phone)
2. On ramp from bank, off ramp to bank
3. Support transfers via phone number/name
4. Support scanning a QR code for transferring to merchants

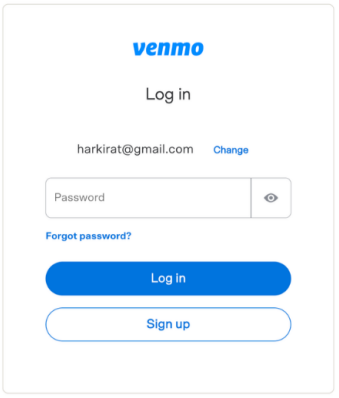
Merchant login

1. Login with google
2. Generate a QR Code for acceptance
3. Merchants get an alert/notification on payment
4. Merchant gets money offramped to bank every 2 days

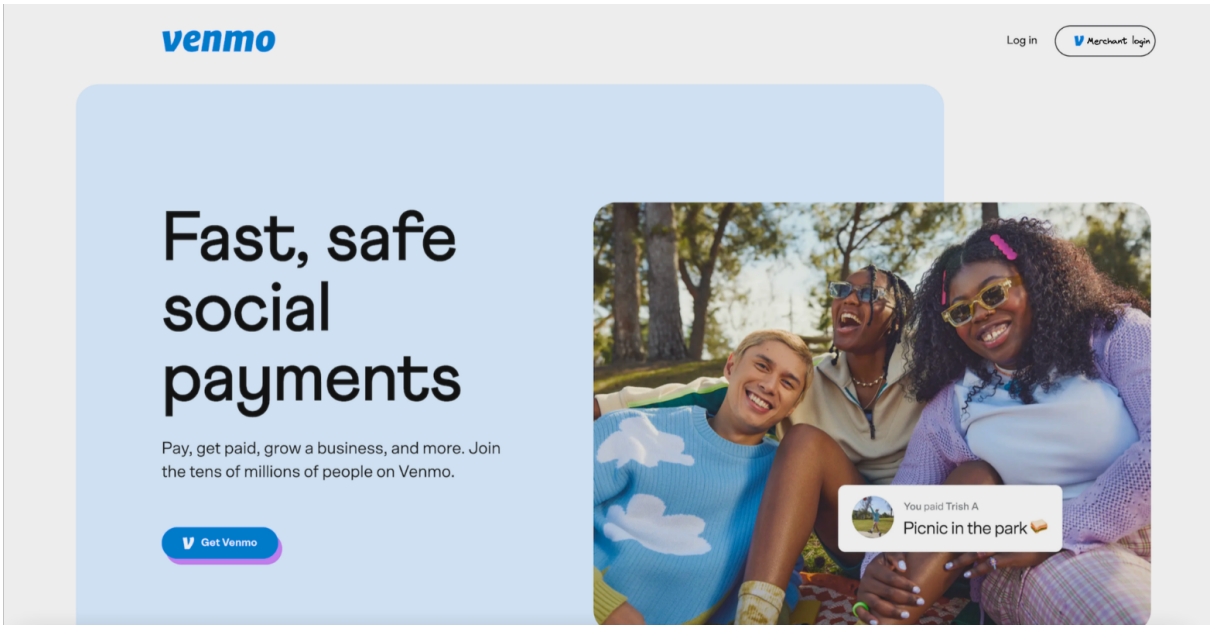
UI/UX (End User)

Login

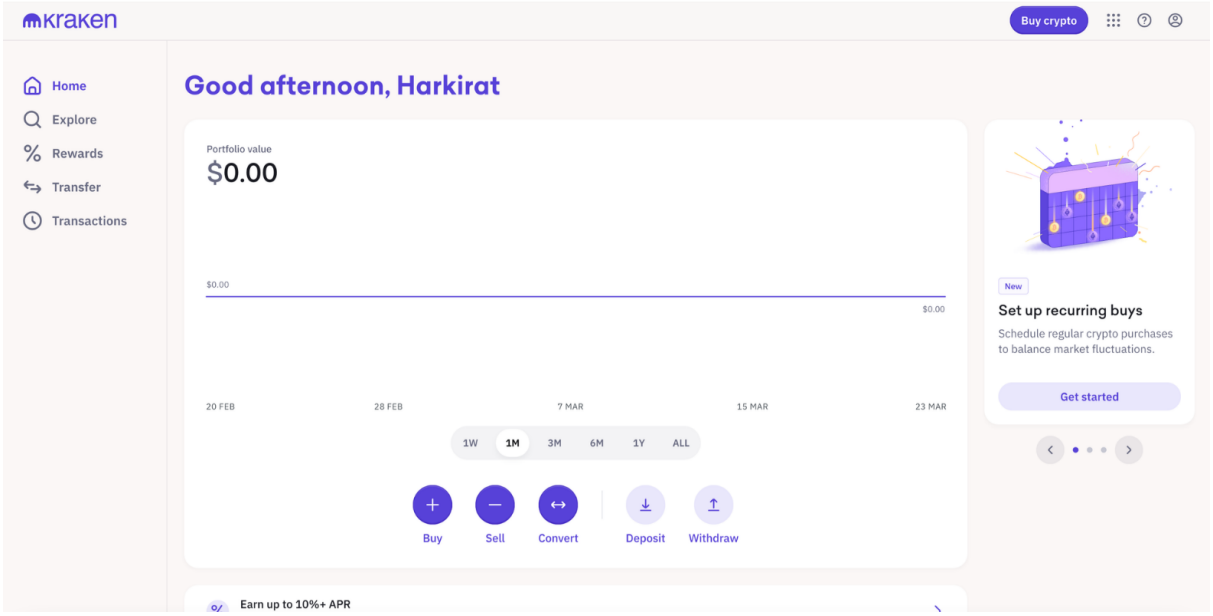




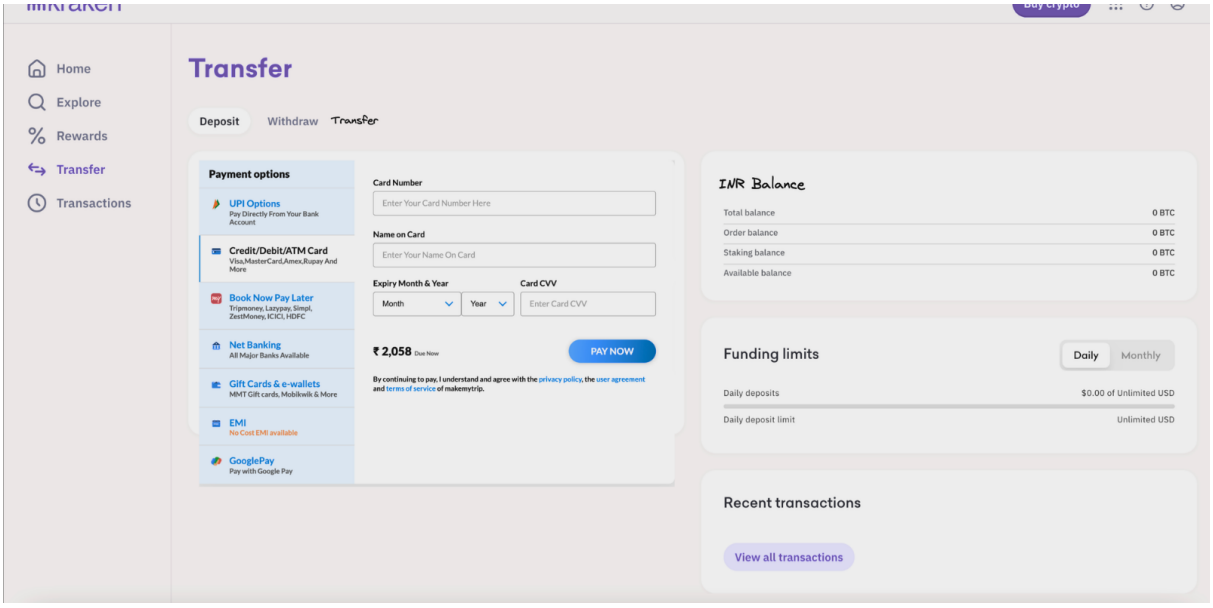
Landing Page

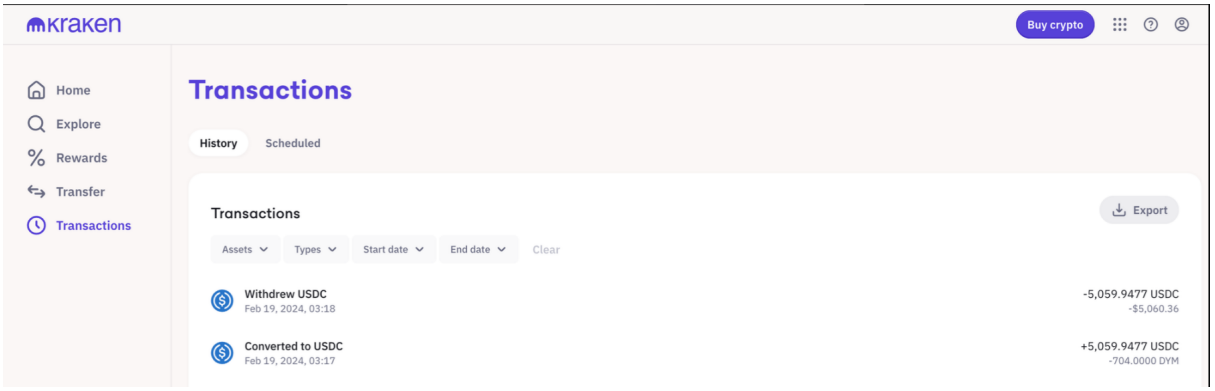


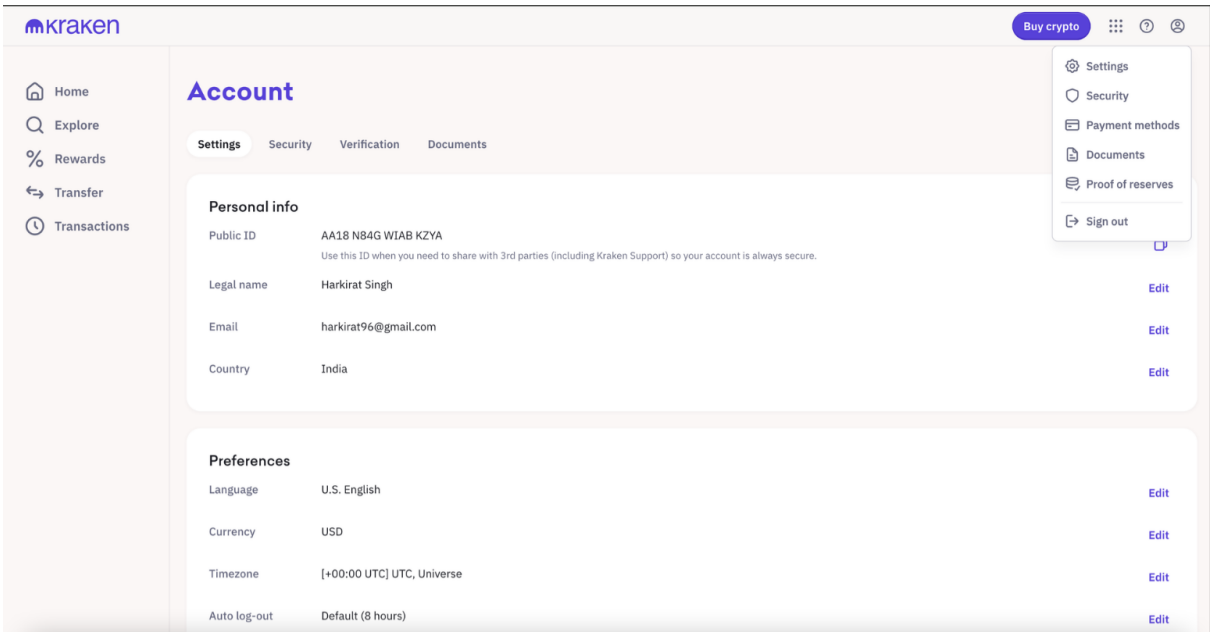
User Home Page



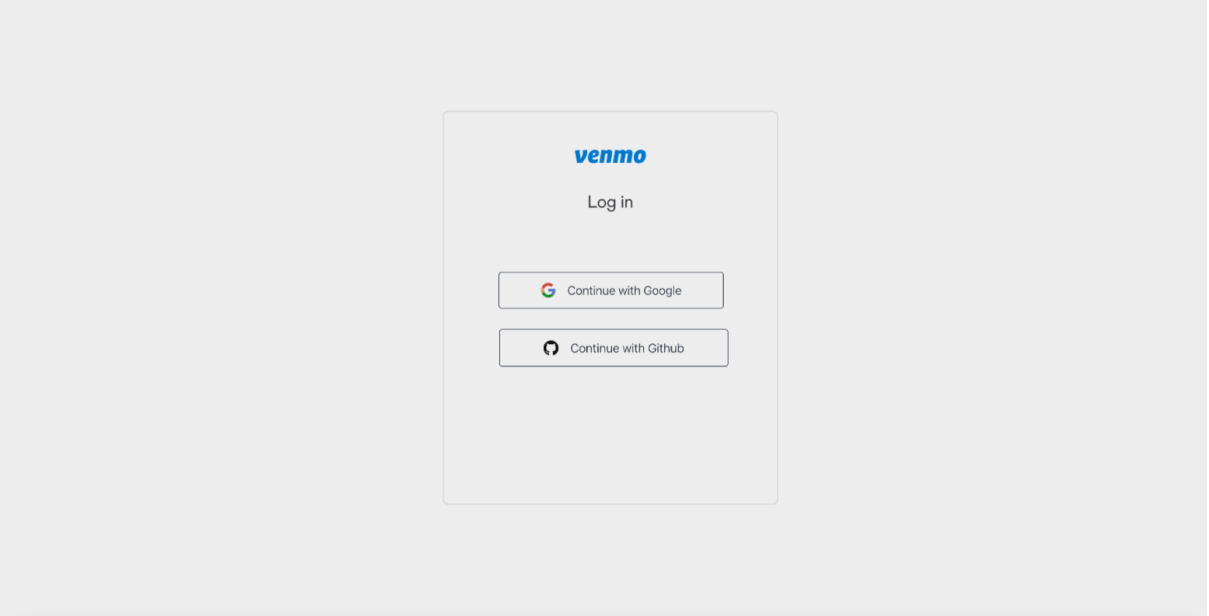
User Transfer Page

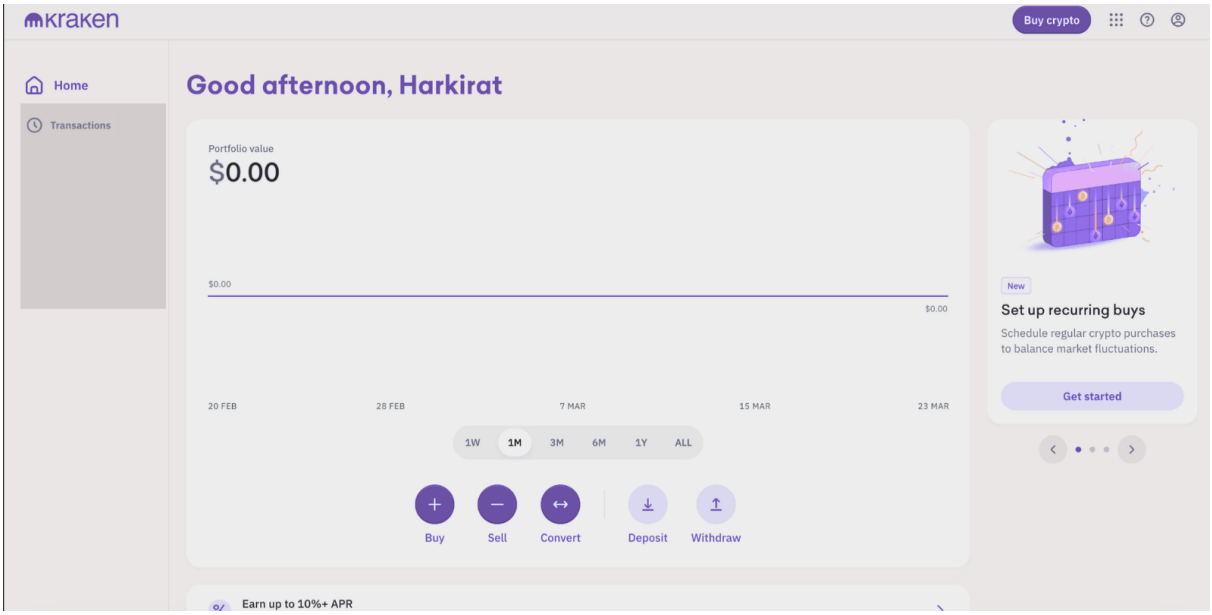


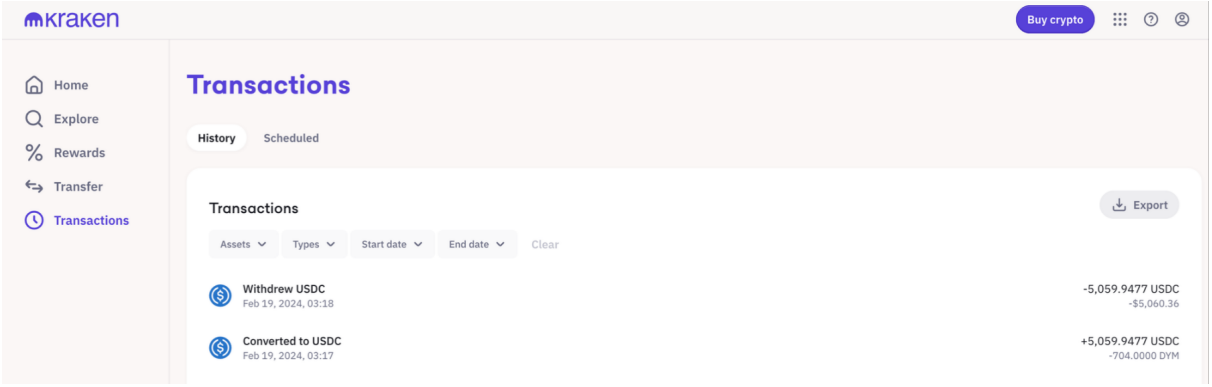




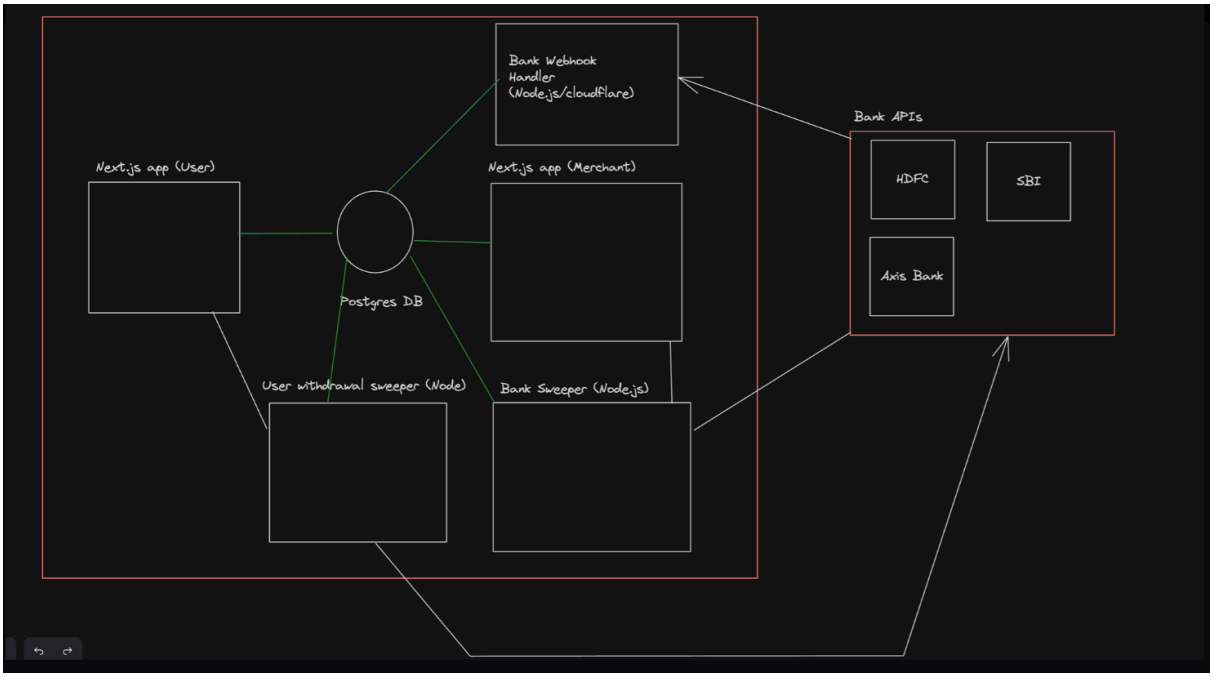
UI/UX (Merchant)

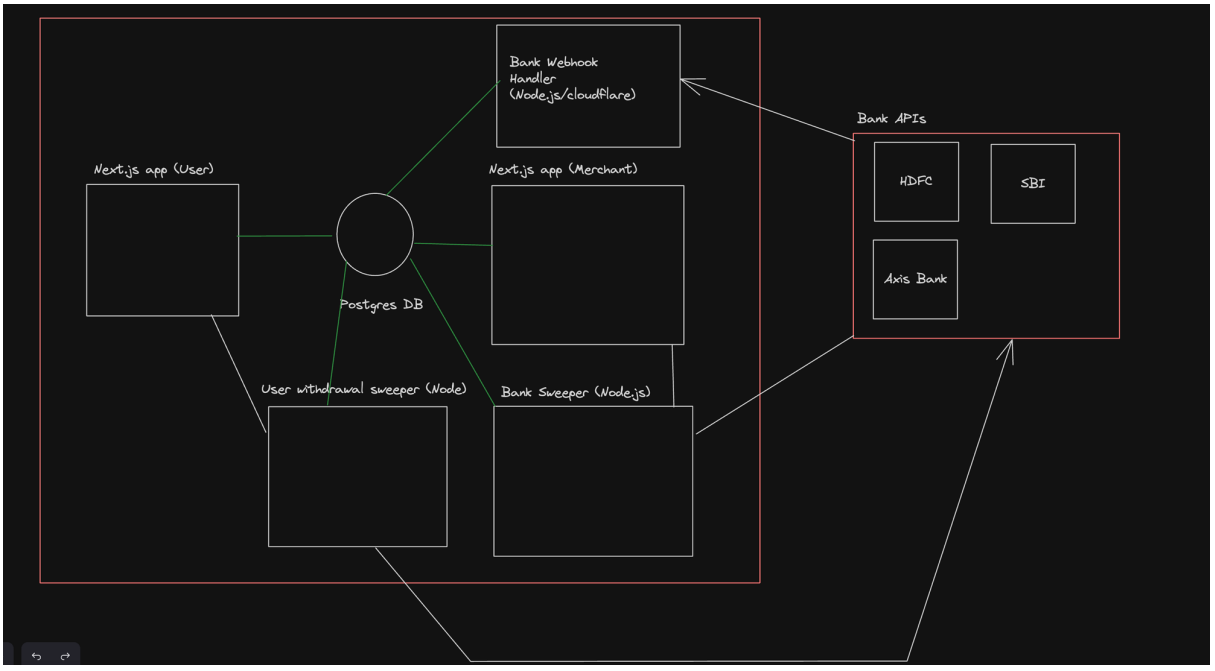

venmo
Log in
Continue with Google
Continue with Github


kraken
Buy crypto
Home
Transactions
Good afternoon, Harkirat
Portfolio value
$0.00
$0.00
$0.00
20 FEB
28 FEB
7 MAR
15 MAR
23 MAR
1W
1M
3M
6M
1Y
ALL
Buy
Sell
Convert
Deposit
Withdraw
New
Set up recurring buys
Schedule regular crypto purchases to balance market fluctuations.
Get started
Earn up to 10%+ APR



Architecture





Hot paths

1. Send money to someone
2. Withdraw balance of merchant
3. Withdraw balance of user back to bank
4. Webhooks from banks to transfer in money

This is ~1 month job for a 2 engineer team.

We can cut scope in either

1. UI
2. Number of features we support (remove merchant altogether)
3. Number of services we need (merge bank server, do withdrawals directly and not in a queue assuming banks are always up)

Stack

1. Frontend and Backend - Next.js (or Backend)
2. Express - Auxilary backends
3. Turborepo
4. Postgres Database

5. Prisma ORM
6. Tailwind

Bootstrap the app

- Init turborepo

```
npx create-turbo@latest
```

- Rename the two Next apps to
  1. user-app
  2. merchant-app
- Add tailwind to it.

```
cd apps/user-app
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p

cd ../merchant-app
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

You can also use https://github.com/vercel/turbo/tree/main/examples/with-tailwind

Update `tailwind.config.js`

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
```

```
    "./app/**/*.{js,ts,jsx,tsx,mdx}",
    "./pages/**/*.{js,ts,jsx,tsx,mdx}",
    "./components/**/*.{js,ts,jsx,tsx,mdx}",
    "../../packages/ui/**/*.{js,ts,jsx,tsx,mdx}"
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Update `global.css`

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

💡 Ensure tailwind is working as expected

Adding prisma

Ref - https://turbo.build/repo/docs/handbook/tools/prisma

1. Create a new `db` folder
2. Initialise package.json

```
npm init -y
npx tsc --init
```

1. Update package.json

```
{
    "name": "@repo/db",
    "version": "0.0.0",
    "dependencies": {
        "@prisma/client": "^5.11.0"
    },
    "devDependencies": {
        "prisma": "5.11.0"
    },
    "exports": {
        "./client": "./index.ts"
    }
}
```

1. Update tsconfig.json

```
{
    "extends": "@repo/typescript-config/react-library.jso
n",
    "compilerOptions": {
      "outDir": "dist"
    },
    "include": ["src"],
    "exclude": ["node_modules", "dist"]
  }
```

1. Init prisma

```
npx prisma init
```

1. Start DB locally/on neon.db/on aiven
2. Update .env with the new database URL
3. Add a basic schema

```
model User {
  id    Int     @id @default(autoincrement())
  email String  @unique
  name  String?
}
```

1. MIgrate DB

```
npx prisma migrate
```

1. Update `index.ts`

```
export * from '@prisma/client';
```

1. Try adding `api/user/route.ts`

```
import { NextResponse } from "next/server"
import { PrismaClient } from "@repo/db/client";

const client = new PrismaClient();

export const GET = async () => {
    await client.user.create({
        data: {
            email: "asd",
            name: "adsads"
        }
    })
    return NextResponse.json({
        message: "hi there"
    })
}
```

Add a recoil/store module

- Create `store` package

```
cd packages
mkdir store
npm init -y
npx tsc --init
```

- Install dependencies

```
npm i recoil
```

- Update tsconfig.json

```
{
    "extends": "@repo/typescript-config/react-library.jso
n",
    "compilerOptions": {
      "outDir": "dist"
    },
    "include": ["src"],
    "exclude": ["node_modules", "dist"]
}
```

- Update package.json

```
{
  "name": "@repo/store",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
```

```
  "dependencies": {
    "recoil": "^0.7.7"
  }
}
```

- Create a simple `atom` called balance in `src/atoms/balance.ts`

```
import { atom } from "recoil";

export const balanceAtom = atom<number>({
    key: "balance",
    default: 0,
})
```

- Create a simple `hook` called `src/hooks/useBalance.ts`

```
import { useRecoilValue } from "recoil"
import { balanceAtom } from "../atoms/balance"

export const useBalance = () => {
    const value = useRecoilValue(balanceAtom);
    return value;
}
```

- Add export to package.json

```
"exports": {
    "./useBalance": "./src/hooks/useBalance"
}
```

Import recoil in the next.js apps

- Install recoil in them

```
npm i recoil
```

- Add a `providers.tsx` file

```
"use client"
import { RecoilRoot } from "recoil";

export const Providers = ({children}: {children: React.Reac
tNode}) => {
    return <RecoilRoot>
        {children}
    </RecoilRoot>
}
```

- Update `layout.tsx`

```
 return (
    <html lang="en">
      <Providers>
        <body className={inter.className}>{children}</body>
      </Providers>
    </html>
  );
```

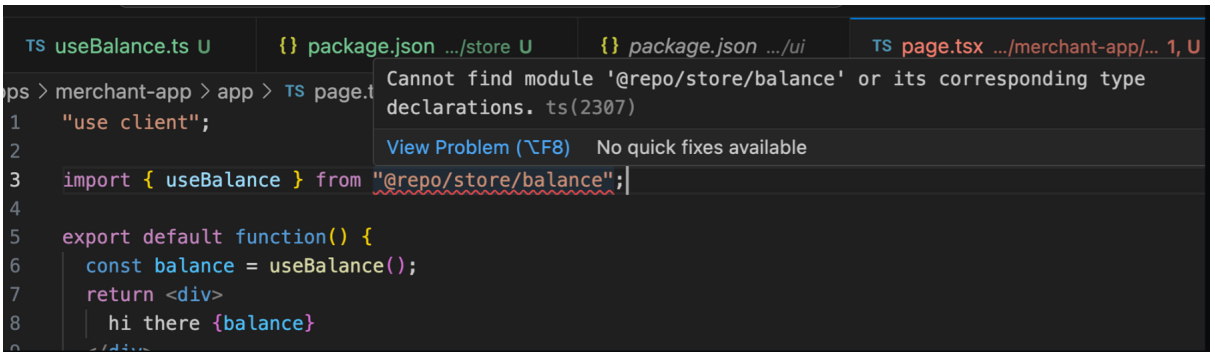
- Create a simple client component and try using the `useBalance` hook in there

```
"use client";

import { useBalance } from "@repo/store/useBalance";

export default function() {
  const balance = useBalance();
  return <div>
    hi there {balance}
  </div>
}
```

If you see this, we'll try to debug it end of class. Should still work in dev mode



Add next-auth

Database

Update the database schema

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id          Int      @id @default(autoincrement())
  email       String?  @unique
  name        String?
  number      String  @unique
  password    String
}

model Merchant {
  id          Int     @id @default(autoincrement())
```

```
  email       String  @unique
  name        String?
  auth_type   AuthType
}

enum AuthType {
  Google
  Github
}
```

User-app

- Go to `apps/user-app`
- Initialize next-auth

```
npm install next-auth
```

- Initialize a simple next auth config in `lib/auth.ts`

```
import db from "@repo/db/client";
import CredentialsProvider from "next-auth/providers/creden
tials"
import bcrypt from "bcrypt";

export const authOptions = {
    providers: [
      CredentialsProvider({
          name: 'Credentials',
          credentials: {
            phone: { label: "Phone number", type: "text", p
laceholder: "1231231231" },
            password: { label: "Password", type: "password"
}
          },
          // TODO: User credentials type from next-aut
          async authorize(credentials: any) {
            // Do zod validation, OTP validation here
            const hashedPassword = await bcrypt.hash(creden
```

```
tials.password, 10);
            const existingUser = await db.user.findFirst({
                where: {
                    number: credentials.phone
                }
            });

            if (existingUser) {
                const passwordValidation = await bcrypt.com
pare(credentials.password, existingUser.password);
                if (passwordValidation) {
                    return {
                        id: existingUser.id.toString(),
                        name: existingUser.name,
                        email: existingUser.number
                    }
                }
                return null;
            }

            try {
                const user = await db.user.create({
                    data: {
                        number: credentials.phone,
                        password: hashedPassword
                    }
                });

                return {
                    id: user.id.toString(),
                    name: user.name,
                    email: user.number
                }
            } catch(e) {
                console.error(e);
            }

            return null
```

```
            },
        })
    ],
    secret: process.env.JWT_SECRET || "secret",
    callbacks: {
        // TODO: can u fix the type here? Using any is bad
        async session({ token, session }: any) {
            session.user.id = token.sub

            return session
        }
    }
}
```

- Create a `/api/auth/[...nextauth]/route.ts`

```
import NextAuth from "next-auth"

const handler = NextAuth(authOptions)

export { handler as GET, handler as POST }
```

- Update .env

```
JWT_SECRET=test
NEXTAUTH_URL=http://localhost:3001
```

- Ensure u see a signin page at http://localhost:3000/api/auth/signin

Merchant-app

Create `lib/auth.ts`

```
import GoogleProvider from "next-auth/providers/google";

export const authOptions = {
    providers: [
        GoogleProvider({
            clientId: process.env.GOOGLE_CLIENT_ID || "",
```

```
            clientSecret: process.env.GOOGLE_CLIENT_SECRET
|| ""
        })
    ],
  }
```

- Create a `/api/auth/[...nextauth]/route.ts`

```
import NextAuth from "next-auth"

const handler = NextAuth(authOptions)

export { handler as GET, handler as POST }
```

- Put your google client and secret in `.env` of the merchant app.
  Ref
  https://next-auth.js.org/providers/google

```
GOOGLE_CLIENT_ID=
GOOGLE_CLIENT_SECRET=
NEXTAUTH_URL=http://localhost:3000
NEXTAUTH_SECRET=kiratsecr3t
```

- Ensure u see a signin page at http://localhost:3001/api/auth/signin
- Try signing in and make sure it reaches the DB

Add auth

Client side

- Wrap the apps around `SessionProvider` context from the next-auth package
- Go to `merchant-app/providers.tsx`

```
"use client"
import { RecoilRoot } from "recoil";
import { SessionProvider } from "next-auth/react";

export const Providers = ({children}: {children: React.Reac
tNode}) => {
    return <RecoilRoot>
        <SessionProvider>
            {children}
        </SessionProvider>
    </RecoilRoot>
}
```

- Do the same for `user-app/providers.tsx`

Server side

Create `apps/user-app/app/api/user.route.ts`

```
import { getServerSession } from "next-auth"
import { NextResponse } from "next/server";
import { authOptions } from "../../lib/auth";

export const GET = async () => {
    const session = await getServerSession(authOptions);
    if (session.user) {
        return NextResponse.json({
            user: session.user
        })
    }
    return NextResponse.json({
        message: "You are not logged in"
    }, {
        status: 403
    })
}
```

Ensure login works as exptected

Add an Appbar component

- Update the `Button` component in UI

```
"use client";

import { ReactNode } from "react";

interface ButtonProps {
  children: ReactNode;
  onClick: () => void;
}

export const Button = ({ onClick, children }: ButtonProps)
=> {
  return (
    <button onClick={onClick} type="button" className="text
-white bg-gray-800 hover:bg-gray-900 focus:outline-none foc
us:ring-4 focus:ring-gray-300 font-medium rounded-lg text-s
m px-5 py-2.5 me-2 mb-2">
      {children}
    </button>

  );
};
```

- Create a `ui/Appbar` component that is highly generic (doesnt know anything about the user/how to logout).

```
import { Button } from "./button";

interface AppbarProps {
    user?: {
        name?: string | null;
    },
    // TODO: can u figure out what the type should be here?
```

```
    onSignin: any,
    onSignout: any
}

export const Appbar = ({
    user,
    onSignin,
    onSignout
}: AppbarProps) => {
    return <div className="flex justify-between border-b px
-4">
        <div className="text-lg flex flex-col justify-cente
r">
            PayTM
        </div>
        <div className="flex flex-col justify-center pt-2">
            <Button onClick={user ? onSignout : onSignin}>
{user ? "Logout" : "Login"}</Button>
        </div>
    </div>
}
```

Checkpoint

We are here - https://github.com/100xdevs-cohort-2/paytm-project-starter-monorepo