

# Building any project

 Akash Kadlag OP Today at 9:18 PM  
Hi @kiratsingh,

Super Excited to build End-To-End App with pretty much everything we learnt in last 16 weeks.

I already DM you the Notion Docs screenshots about Features of Projects.  
But Here I wanted to share something important...

There are many videos available about build XYZ.  
Even you also have Code With Me type Videos.  
These videos are good.

The problem starts when we try to build this kind of app by ourself.  
Here are few challenges

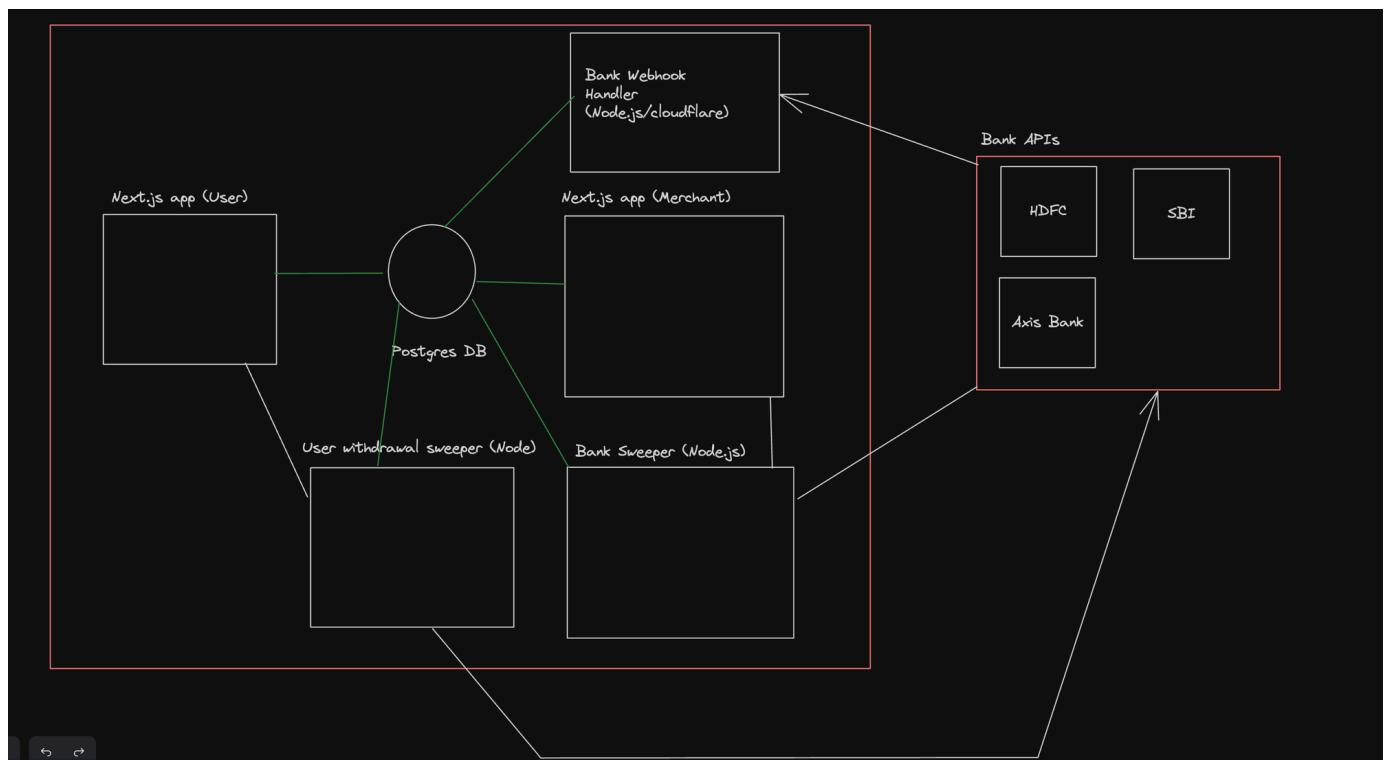
1. Where to Start...?
2. Design UI/UX
3. Low Level Design
4. High Level Design
5. ER Diagram
6. Tech Stack
7. How to think about new Features.
8. How much complexity is needed..?

## Points

1. Where to start - Feature planning
2. Design UI/UX
  1. UX - First principles/Copy the biggest website out there
  2. UI - Designer. Today there are tools but havent found any good one
3. High level Design
  1. Auth provider
  2. Database
  3. Backend Stack
  4. Frontend stack
  5. Modules you'll have (common/ui/backend)
  6. Cloud to deploy to

## 4. LLD

1. Schema
2. Route signatures
3. Frontend Components - debatable
5. ER Diagrams -
  1. We can build these today, but usually not needed unless you're a very visual person
6. How to think about features
  1. Usually come from product
  2. If you're a founder, then just whatever u think is right
7. How much complexity is needed
  1. Depends on the size of the company. For a startup, whatever helps you move fast w/o tech debt. For a company there are a lot of layers of review to go through



**💡** If you want a voice speaker that says **Paytm Received 100 rs from someone** we need to add another ws service in here  
We'll also be doing DB polling in the middle which should rather be done via a queue

# 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

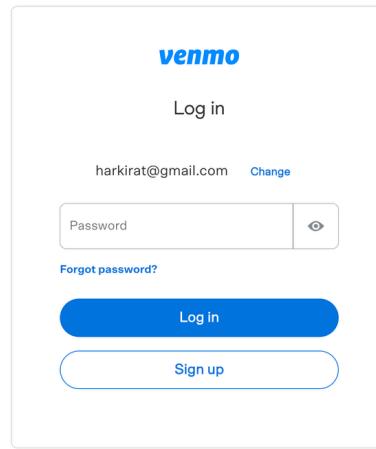
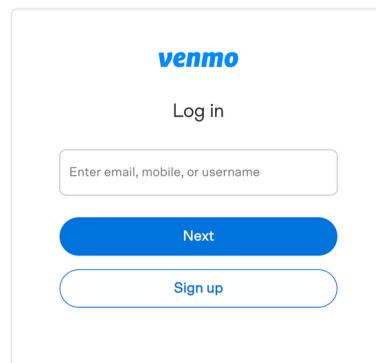
## Merchant login

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

**Transfer**

**Payment options**

- UPI Options** Pay Directly From Your Bank Account
- Credit/Debit/ATM Card** Visa, MasterCard, Amex, Rupay And More
- Book Now Pay Later** Tripromoney, Lazypay, Simpl, ZestMoney, ICICI, HDFC
- Net Banking** All Major Banks Available
- Gift Cards & e-wallets** MMT Gift cards, Mobikwik & More
- EMI** No Cost EMI available
- GooglePay** Pay with Google Pay

**INR Balance**

Total balance	0 BTC
Order balance	0 BTC
Staking balance	0 BTC
Available balance	0 BTC

**Funding limits**

Daily	Monthly
\$0.00 of Unlimited USD	Unlimited USD
Daily deposit limit	

**Recent transactions**

[View all transactions](#)

**Transactions**

**History** Scheduled

**Transactions**

**Export**

Type	Date	Amount
Withdrew USDC	Feb 19, 2024, 03:18	-5,059.9477 USDC -\$5,060.36
Converted to USDC	Feb 19, 2024, 03:17	+5,059.9477 USDC -704.0000 DYM

**Account**

**Settings** Security Verification Documents

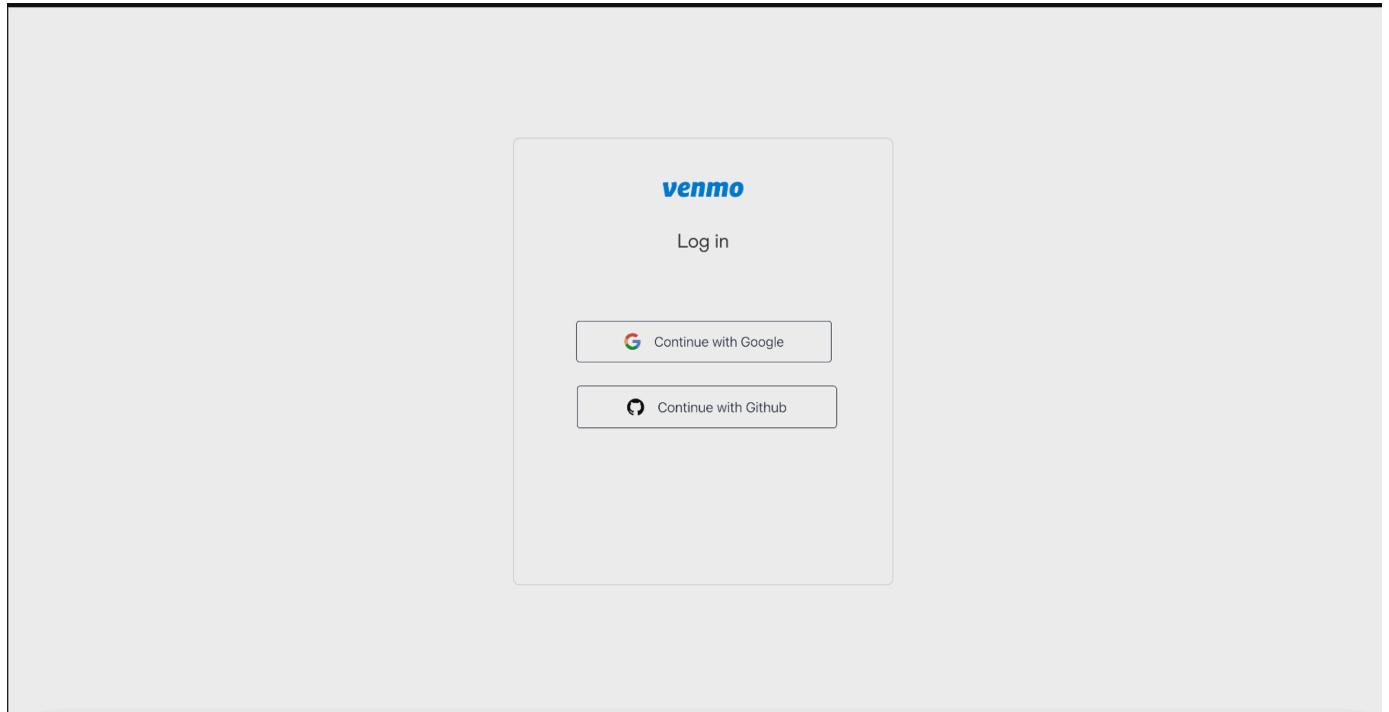
**Personal info**

Public ID	AA18 N84G WIAB KZYA
Use this ID when you need to share with 3rd parties (including Kraken Support) so your account is always secure.	
Legal name	Harkirat Singh
Email	harkirat96@gmail.com
Country	India

**Settings**

- Settings
- Security
- Payment methods
- Documents
- Proof of reserves
- Sign out

# UI/UX (Merchant)



**kraken**

Good afternoon, Harkirat

Portfolio value  
\$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

Earn up to 10%+ APR

New Set up recurring buys Schedule regular crypto purchases to balance market fluctuations. Get started

A screenshot of the Kraken home page. It displays a portfolio value of \$0.00. Below the value is a timeline from February 20 to March 23. A series of buttons for 'Buy', 'Sell', 'Convert', 'Deposit', and 'Withdraw' are shown. At the bottom, there's a note about earning up to 10%+ APR. On the right side, there's a section titled 'Set up recurring buys' with a 'Get started' button.

**kraken**

Transactions

History Scheduled

Transactions

Assets Types Start date End date Clear

Withdrew USDC Feb 19, 2024, 03:18 -5,059.9477 USDC -\$5,060.36

Converted to USDC Feb 19, 2024, 03:17 +5,059.9477 USDC -704.0000 DYM

Export

A screenshot of the Kraken transactions page. It shows a history of transactions, including a withdrawal of USDC and a conversion to USDC. There are filters for assets, types, start date, end date, and clear. An 'Export' button is located in the top right corner.

# Architecture



# 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;      Copy
@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          Copy
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.json",
    "compilerOptions": {
        "outDir": "dist"
    },
    "include": ["src"],
    "exclude": ["node_modules", "dist"]
}

```

[Copy](#)

## 1. Init prisma

```
npx prisma
```

[Copy](#)

## 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?
}

```

[Copy](#)

## 1. Migrate DB

```
npx prisma migrate
```

[Copy](#)

## 1. Update index.ts

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

Copy

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"
    })
}
```

Copy

## Add a recoil/store module

- Create `store` package

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

Copy

- Install dependencies

```
Copy
npm i recoil
```

- Update tsconfig.json

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

- Update package.json

```
Copy
{
  "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`

```
Copy
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"          Copy
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          Copy
```

- Add a `providers.tsx` file

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

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

- Update `layout.tsx`

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

Copy

- 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>
}
```

Copy

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"
```

Copy

```

}

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/credentials"
import bcrypt from "bcrypt";

export const authOptions = {
  providers: [
    CredentialsProvider({

```

[Copy](#)

```
name: 'Credentials',
credentials: {
  phone: { label: "Phone number", type: "text", placeholder: " "
  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(credentials.password, 10);
  const existingUser = await db.user.findFirst({
    where: {
      number: credentials.phone
    }
  });

  if (existingUser) {
    const passwordValidation = await bcrypt.compare(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"          Copy

const handler = NextAuth(authOptions)

export { handler as GET, handler as POST }

```

- Update .env

```

JWT_SECRET=test          Copy
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";          Copy

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 }
```

[Copy](#)

- 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
```

[Copy](#)

- 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";
```

[Copy](#)

```
export const Providers = ({children}: {children: React.ReactNode}) => {
  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
  })
}
```

Copy

Ensure login works as expected

# 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-gradual">
      {children}
    </button>

  );
};
```

[Copy](#)

- 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">
```

[Copy](#)

```
<div className="text-lg flex flex-col justify-center">
  PayTM
</div>
<div className="flex flex-col justify-center pt-2">
  <Button onClick={user ? onSignout : onSignin}>{user ? "Logou
</div>
</div>
}
```

# Checkpoint

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

# On Ramping

## Creating a dummy bank server

- Allows PayTM to generate a `token` for a payment for a user for some amount

```
POST /api/transaction
{
  "user_identifier": "1",
  "amount": "59900", // Rs 599
  "webhookUrl": "http://localhost:3003/hdfcWebhook"
}
```

Copy

- PayTM should redirect the user to

```
https://bank-api-frontend.com/pay?token={token\_from\_step\_1
```

Copy

- If user made a successful payment, `Bank` should hit the `webhookUrl` for the company

## Creating a bank\_webhook\_handler Node.js project

- Init node.js project + esbuild

```
cd apps
mkdir bank_webhook_handler
cd bank_webhook_handler
npm init -y
npx tsc --init
npm i esbuild express @types/express
```

[Copy](#)

- Update tsconfig

```
{
  "extends": "@repo/typescript-config/base.json",
  "compilerOptions": {
    "outDir": "dist"
  },
  "include": ["src"],
  "exclude": ["node_modules", "dist"]
}
```

[Copy](#)

- Create `src/index.ts`

```
import express from "express";
const app = express();

app.post("/hdfcWebhook", (req, res) => {
  //TODO: Add zod validation here?
  const paymentInformation = {
    token: req.body.token,
    userId: req.body.user_identifier,
    amount: req.body.amount
  };
  // Update balance in db, add txn
})
```

[Copy](#)

- Update DB Schema

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

datasource db {
```

[Copy](#)

```
provider = "postgresql"
url      = env("DATABASE_URL")
}

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

model Merchant {
    id          Int      @id @default(autoincrement())
    email       String   @unique
    name        String?
    auth_type   AuthType
}

model OnRampTransaction {
    id          Int      @id @default(autoincrement())
    status      OnRampStatus
    token       String   @unique
    provider    String
    amount      Int
    startTime   DateTime
    userId      Int
    user        User     @relation(fields: [userId], references: [id])
}

model Balance {
    id          Int      @id @default(autoincrement())
    userId      Int      @unique
    amount      Int
    locked      Int
    user        User     @relation(fields: [userId], references: [id])
}

enum AuthType {
    Google
    Github
}

enum OnRampStatus {
```

```
Success
Failure
Processing
}
```

- Migrate the DB

Go to the right folder (packages/db) Copy  
 npx prisma migrate dev --name add\_balance

- Add `repo/db` as a dependency to package.json

"@repo/db": Copy

- Add transaction to update the balance and transactions DB

Ref - <https://www.prisma.io/docs/orm/prisma-client/queries/transactions>

```
import express from "express";
import db from "@repo/db/client";
const app = express();

app.use(express.json())

app.post("/hdfcWebhook", async (req, res) => {
  //TODO: Add zod validation here?
  //TODO: HDFC bank should ideally send us a secret so we know this is
  const paymentInformation: {
    token: string;
    userId: string;
    amount: string
  } = {
    token: req.body.token,
    userId: req.body.user_identifier,
    amount: req.body.amount
  };

  try {
    await db.$transaction([
      db.balance.updateMany({
        where: {
          userId: Number(paymentInformation.userId)
        },
        data: {

```

```
        amount: {
            // You can also get this from your DB
            increment: Number(paymentInformation.amount)
        }
    },
}),
db.onRampTransaction.updateMany({
    where: {
        token: paymentInformation.token
    },
    data: {
        status: "Success",
    }
})
]);
}

res.json({
    message: "Captured"
})
} catch(e) {
    console.error(e);
    res.status(411).json({
        message: "Error while processing webhook"
    })
}
}

app.listen(3003);
```

## Create generic appbar

- Create a new `AppbarClient` component in `apps/user-app/AppbarClient.tsx`

```
"use client"

import { signIn, signOut, useSession } from "next-auth/react";
import { Appbar } from "@repo/ui/appbar";
import { useRouter } from "next/navigation";

export function AppbarClient() {
  const session = useSession();
  const router = useRouter();

  return (
    <div>
      <Appbar onSignin={signIn} onSignout={async () => {
        await signOut()
        router.push("/api/auth/signin")
      }} user={session.data?.user} />
    </div>
  );
}

}
```

Copy

- Add AppbarClient to `layout.tsx`

```
import "./globals.css";
import type { Metadata } from "next";
import { Inter } from "next/font/google";
import { Providers } from "../provider";
import { AppbarClient } from "../components/AppbarClient";

const inter = Inter({ subsets: ["latin"] });

export const metadata: Metadata = {
  title: "Wallet",
  description: "Simple wallet app",
};

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}): JSX.Element {
  return (
    <html lang="en">
      <Providers>
        <AppbarClient />
        <body className={inter.className}>{children}</body>
    
```

Copy

```
</Providers>
</html>
);
}
```

# Create sidebar

- Create `user-app/(dashboard)` folder
- Add 3 pages inside it
  - `dashboard/page.tsx`
  - `transactions/page.tsx`
  - `transfer/page.tsx`

Copy

```
export default function() {
  return <div>
    Dashboard Page (or transfer/txn page)
  </div>
}
```

- Create `user-app/(dashboard)/layout.tsx`



Icons come from <https://heroicons.com/>  
SidebarItem will be created in the next step

Copy

```
import { SidebarItem } from "../../components/SidebarItem";

export default function Layout({
  children,
}: {
  children: React.ReactNode;
}): JSX.Element {
```

```

        return (
            <div className="flex">
                <div className="w-72 border-r border-slate-300 min-h-screen mr-4">
                    <div>
                        <SidebarItem href="/dashboard" icon={<HomeIcon />} title="Dashboard" />
                        <SidebarItem href="/transfer" icon={<TransferIcon />} title="Transfer" />
                        <SidebarItem href="/transactions" icon={<TransactionsIcon />} title="Transactions" />
                    </div>
                </div>
                {children}
            </div>
        );
    }

// Icons Fetched from https://heroicons.com/
function HomeIcon() {
    return <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24">
        <path stroke-linecap="round" stroke-linejoin="round" d="M2.25 12 8.9 18.5l1.5 1.5m15.75 -12 5.5 5.5l5.5 -5.5m-15.75 12 5.5 5.5l5.5 -5.5" />
    </svg>
}

function TransferIcon() {
    return <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24">
        <path stroke-linecap="round" stroke-linejoin="round" d="M7.5 21 3 16l4.5 -4.5m15.75 -12 5.5 5.5l5.5 -5.5m-15.75 12 5.5 5.5l5.5 -5.5" />
    </svg>
}

function TransactionsIcon() {
    return <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24">
        <path stroke-linecap="round" stroke-linejoin="round" d="M12 6v6h4.5m-4.5 0v6h4.5m-4.5 -6h4.5" />
    </svg>
}

```

- Create `SidebarItem` component

```

"use client"

import { usePathname, useRouter } from "next/navigation";
import React from "react";

export const SidebarItem = ({ href, title, icon }: { href: string; title: string; icon: React.ComponentType }) => {
    const router = useRouter();
    const pathname = usePathname()
    const selected = pathname === href

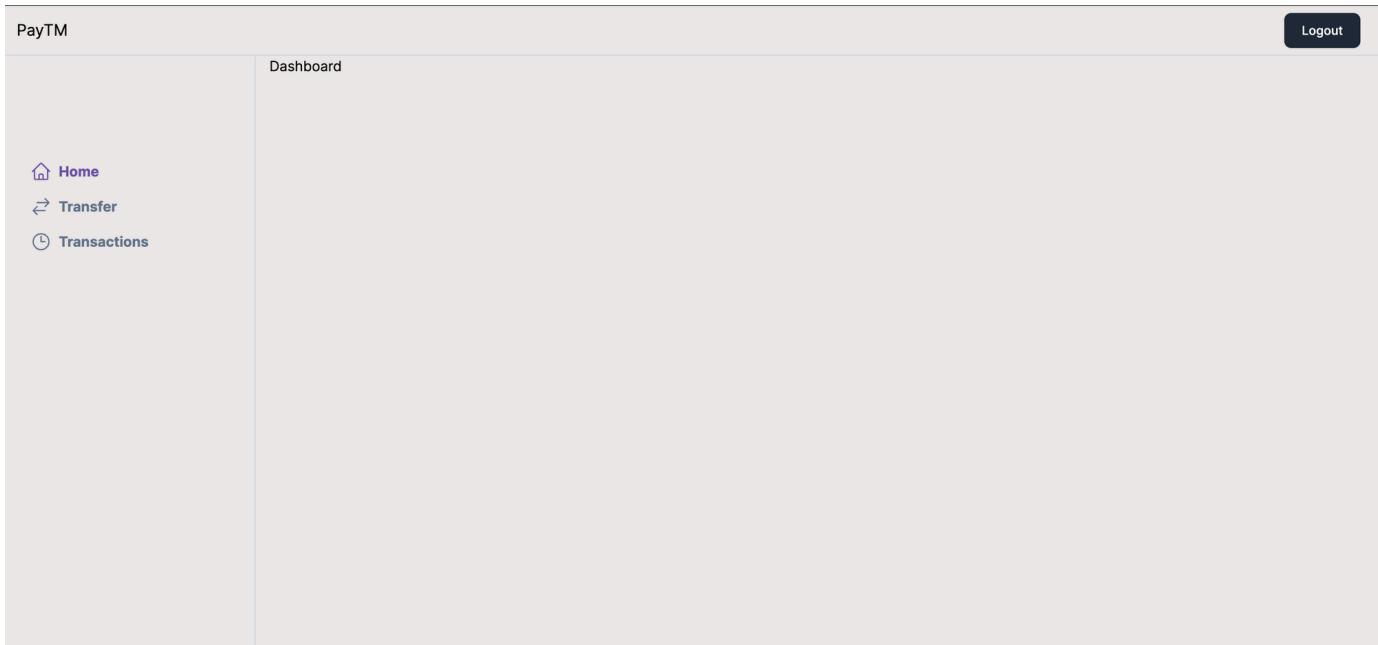
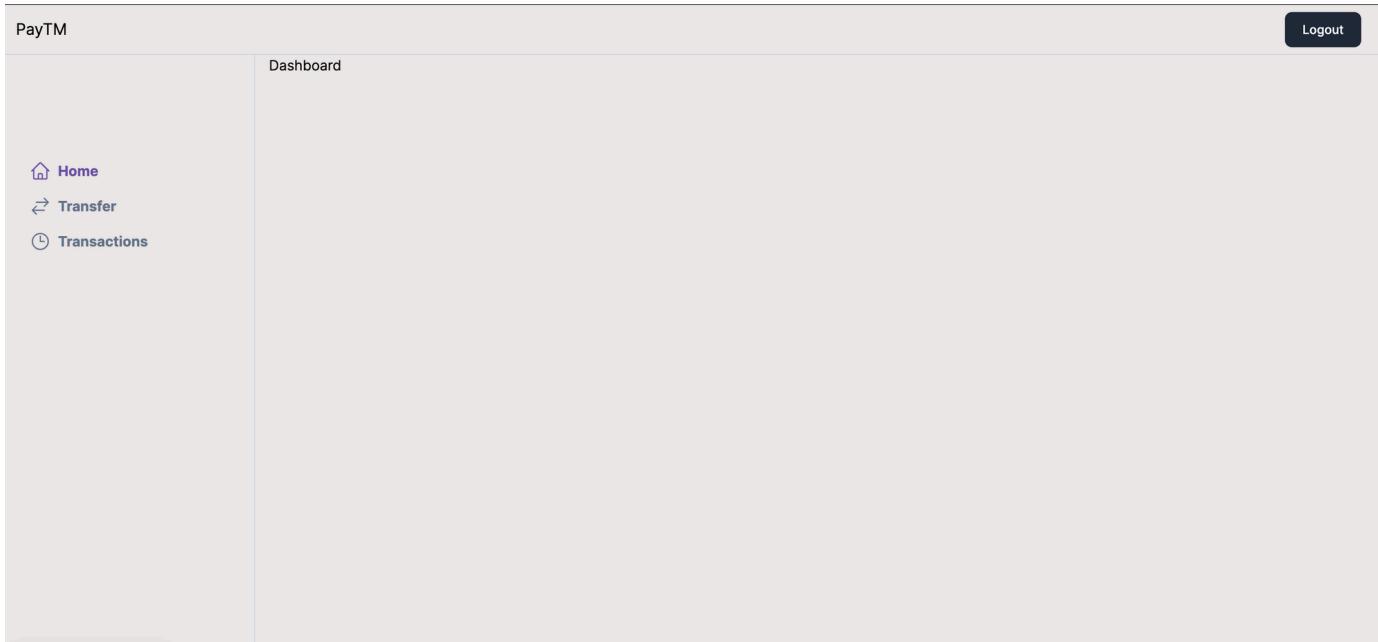
    return <div className={`flex ${selected ? "text-[#6a51a6]" : "text-slate-500"}`}>
        <div style={{ flex: 1 }}>
            <span>{title}</span>
            {icon}
        </div>
        <div style={{ flex: 1 }}>
            <a href={href}>{title}</a>
        </div>
    </div>
}

```

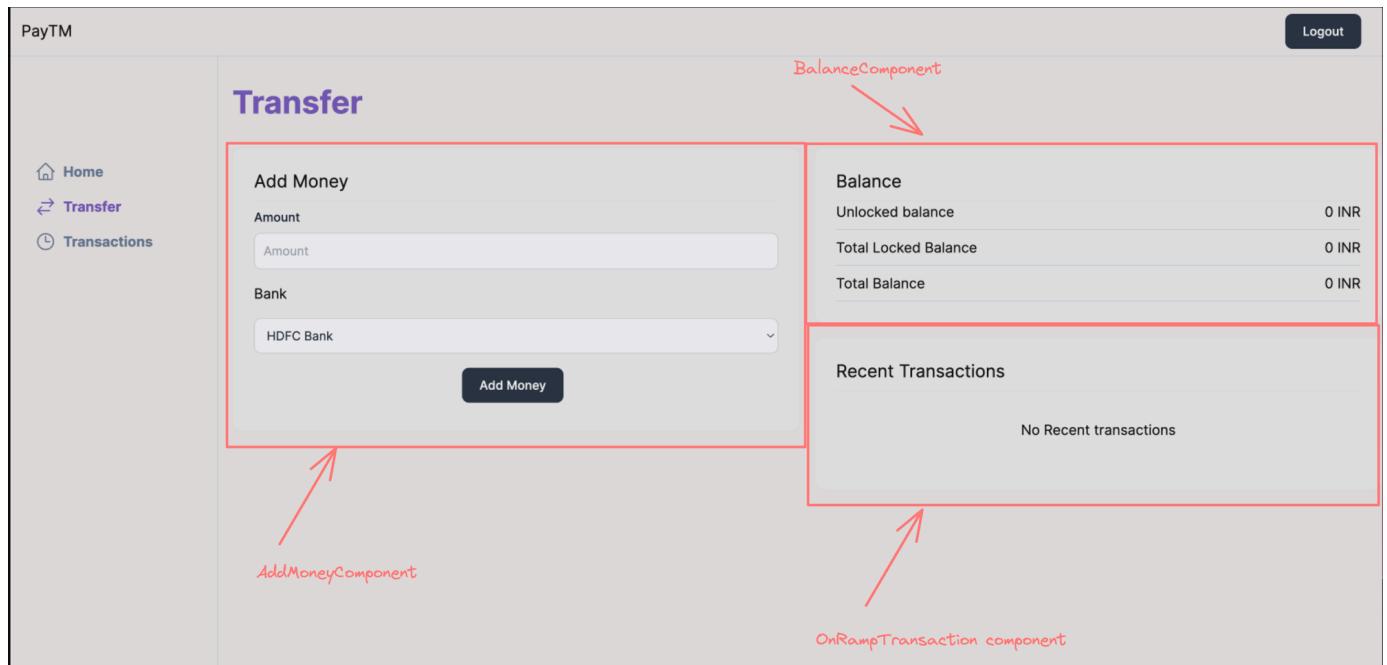
Copy

```
        router.push(href);  
    } }>  
    <div className="pr-2">  
        {icon}  
    </div>  
    <div className={`font-bold ${selected ? "text-[#6a51a6]" : "text  
        {title}  
    </div>  
    </div>  
}
```

You should have 3 pages that look like this



# Create transfer page



- Add a better `card` component to `packages/ui`

```
import React from "react";
export function Card({
  title,
  children,
}: {
  title: string;
  children?: React.ReactNode;
}): JSX.Element {
  return (
    <div
      className="border p-4"
    >
      <h1 className="text-xl border-b pb-2">
        {title}
      </h1>
      <p>{children}</p>
    </div>
  )
}
```

Copy

```
    );
}
```

- Add a `Center` component that centralizes (both vertically and horizontally) the children given to it (in `packages/ui` )

 Make sure to export it in `package.json`

```
import React from "react"

export const Center = ({ children }: { children: React.ReactNode }) => {
  return <div className="flex justify-center flex-col h-full">
    <div className="flex justify-center">
      {children}
    </div>
  </div>
}
```

Copy

- Add a `Select` component to `packages/ui` (Make sure to export it)

```
"use client"
export const Select = ({ options, onSelect }: {
  onSelect: (value: string) => void;
  options: {
    key: string;
    value: string;
  }[];
}) => {
  return <select onChange={(e) => {
    onSelect(e.target.value)
  }} className="bg-gray-50 border border-gray-300 text-gray-900 text-s
    {options.map(option => <option value={option.key}>{option.value}</option>}
  </select>
}
```

Copy

- Add `TextInput` component to `packages/ui`

```
"use client"

export const TextInput = {
  placeholder,
  onChange,
```

Copy

```

    label
  }: {
    placeholder: string;
    onChange: (value: string) => void;
    label: string;
}) => {
  return <div className="pt-2">
    <label className="block mb-2 text-sm font-medium text-gray-900">
      <input onChange={(e) => onChange(e.target.value)} type="text" id
    </div>
}

```

- Create `user-app/components/AddMoneyCard.tsx`

```

"use client"
import { Button } from "@repo/ui/button";
import { Card } from "@repo/ui/card";
import { Center } from "@repo/ui/center";
import { Select } from "@repo/ui/select";
import { useState } from "react";
import { TextInput } from "@repo/ui/textinput";

const SUPPORTED_BANKS = [
  {
    name: "HDFC Bank",
    redirectUrl: "https://netbanking.hdfcbank.com"
  },
  {
    name: "Axis Bank",
    redirectUrl: "https://www.axisbank.com/"
  }
];

export const AddMoney = () => {
  const [redirectUrl, setRedirectUrl] = useState(SUPPORTED_BANKS[0].redirectUrl);
  return <Card title="Add Money">
    <div className="w-full">
      <TextInput label={"Amount"} placeholder={"Amount"} onChange={() =>
        setRedirectUrl(SUPPORTED_BANKS.find(x => x.name === value).redirectUrl)
      } />
      <div className="py-4 text-left">
        Bank
      </div>
      <Select onSelect={(value) => {
        setRedirectUrl(SUPPORTED_BANKS.find(x => x.name === value).redirectUrl)
      }} options={SUPPORTED_BANKS.map(x => ({
        key: x.name,
        value: x.name
      }))}>
    
```

Copy

```

        })} />
      <div className="flex justify-center pt-4">
        <Button onClick={() => {
          window.location.href = redirectUrl || "";
        }}>
          Add Money
        </Button>
      </div>
    </div>
</Card>
}

```

- Create `user-app/components/BalanceCard.tsx`



We're diving my 100 because we store in `paise` denomination in the db

Copy

```

import { Card } from "@repo/ui/card";

export const BalanceCard = ({amount, locked}: {
  amount: number;
  locked: number;
}) => {
  return <Card title={"Balance"}>
    <div className="flex justify-between border-b border-slate-300 p
      <div>
        Unlocked balance
      </div>
      <div>
        {amount / 100} INR
      </div>
    </div>
    <div className="flex justify-between border-b border-slate-300 p
      <div>
        Total Locked Balance
      </div>
      <div>
        {locked / 100} INR
      </div>
    </div>
    <div className="flex justify-between border-b border-slate-300 p
      <div>
        Total Balance
      </div>
      <div>

```

```

        {(locked + amount) / 100} INR
    </div>
</div>
</Card>
}

```

- Create `user-app/components/OnRampTransaction.tsx`

```

import { Card } from "@repo/ui/card" Copy

export const OnRampTransactions = ({
  transactions
}: {
  transactions: {
    time: Date,
    amount: number,
    // TODO: Can the type of `status` be more specific?
    status: string,
    provider: string
  }[]
}) => {
  if (!transactions.length) {
    return <Card title="Recent Transactions">
      <div className="text-center pb-8 pt-8">
        No Recent transactions
      </div>
    </Card>
  }
  return <Card title="Recent Transactions">
    <div className="pt-2">
      {transactions.map(t => <div className="flex justify-between">
        <div>
          <div className="text-sm">
            Received INR
          </div>
          <div className="text-slate-600 text-xs">
            {t.time.toDateString()}
          </div>
        </div>
        <div className="flex flex-col justify-center">
          + Rs {t.amount / 100}
        </div>
      </div>)
    </div>
  </Card>
}

```

```
</Card>
}
```

- Create `user-app/app/(dashboard)/transfer/page.tsx`

Copy

```
import prisma from "@repo/db/client";
import { AddMoney } from "../../components/AddMoneyCard";
import { BalanceCard } from "../../components/BalanceCard";
import { OnRampTransactions } from "../../components/OnRampTransactio
import { getServerSession } from "next-auth";
import { authOptions } from "../../lib/auth";

async function getBalance() {
  const session = await getServerSession(authOptions);
  const balance = await prisma.balance.findFirst({
    where: {
      userId: Number(session?.user?.id)
    }
  });
  return {
    amount: balance?.amount || 0,
    locked: balance?.locked || 0
  }
}

async function getOnRampTransactions() {
  const session = await getServerSession(authOptions);
  const txns = await prisma.onRampTransaction.findMany({
    where: {
      userId: Number(session?.user?.id)
    }
  });
  return txns.map(t => {
    time: t.startTime,
    amount: t.amount,
    status: t.status,
    provider: t.provider
  })
}

export default async function() {
  const balance = await getBalance();
  const transactions = await getOnRampTransactions();

  return <div className="w-screen">
```

```
<div className="text-4xl text-[#6a51a6] pt-8 mb-8 font-bold">
  Transfer
</div>
<div className="grid grid-cols-1 gap-4 md:grid-cols-2 p-4">
  <div>
    <AddMoney />
  </div>
  <div>
    <BalanceCard amount={balance.amount} locked={balance.loc
    <div className="pt-4">
      <OnRampTransactions transactions={transactions} />
    </div>
  </div>
  </div>
</div>
}
```

## Add some seed data

- Go to packages/db
- Add prisma/seed.ts

```
import { PrismaClient } from '@prisma/client'
const prisma = new PrismaClient()

async function main() {
  const alice = await prisma.user.upsert({
    where: { number: '9999999999' },
    update: {},
    create: {
      number: '9999999999',
      password: 'alice',
      name: 'alice',
      OnRampTransaction: {
        create: {
          startTime: new Date(),
          status: "Success",
          amount: 20000,
        }
      }
    }
  })
}

main().catch((e) => {
  console.error(e)
})
```

```

        token: "122",
        provider: "HDFC Bank",
    },
},
},
})
const bob = await prisma.user.upsert({
    where: { number: '9999999998' },
    update: {},
    create: {
        number: '9999999998',
        password: 'bob',
        name: 'bob',
        OnRampTransaction: {
            create: {
                startTime: new Date(),
                status: "Failure",
                amount: 2000,
                token: "123",
                provider: "HDFC Bank",
            },
        },
    },
})
console.log({ alice, bob })
}
main()
.then(async () => {
    await prisma.$disconnect()
})
.catch(async (e) => {
    console.error(e)
    await prisma.$disconnect()
    process.exit(1)
})

```

- Update package.json

```
"prisma": {
    "seed": "ts-node prisma/seed.ts"
}
```

Copy

- Run command to seed db

```
npx prisma db Copy
```

- Explore db

```
npx prisma studio Copy
```

## Make landing page redirect

The user should go to either the `signin` page or the `dashboard` page based on if they are logged in

Update root `page.tsx`

```
import { getServerSession } from "next-auth";
import { redirect } from 'next/navigation'
import { authOptions } from "./lib/auth";

export default async function Page() {
  const session = await getServerSession(authOptions);
  if (session?.user) {
    redirect('/dashboard')
  } else {
    redirect('/api/auth/signin')
  }
}
```

Copy

Final codebase - <https://github.com/100xdevs-cohort-2/week-17-final-code>

