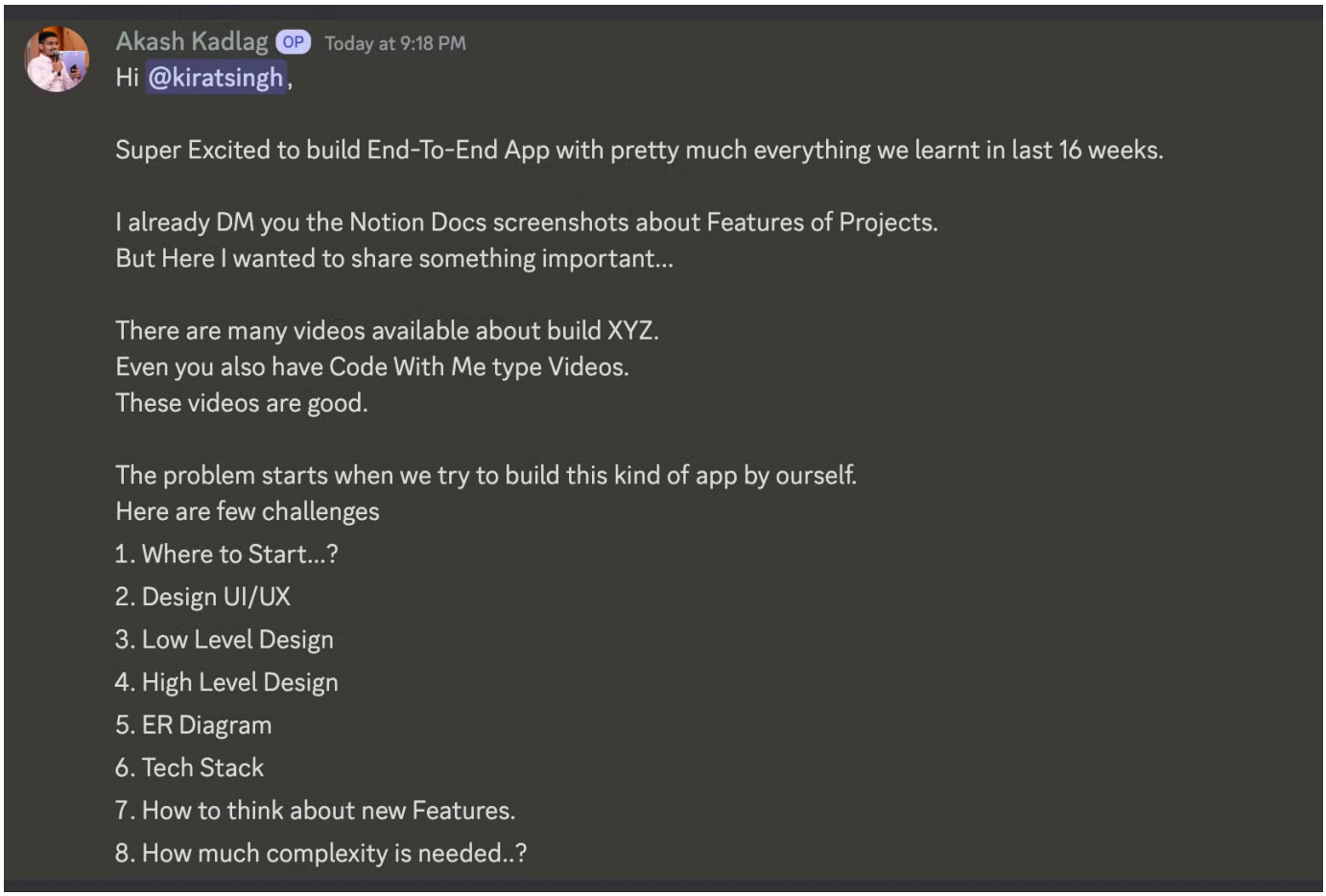
Building any project

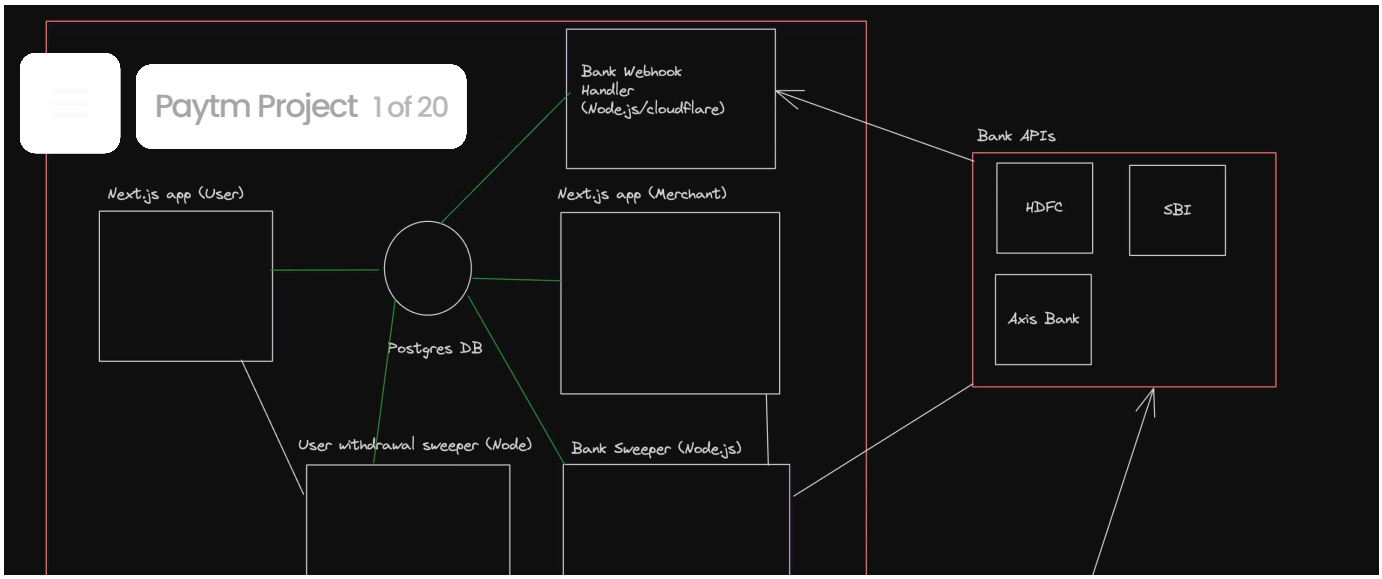


Points

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

4. Frontend stack
5. M (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



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

Paytm Project 1 of 20

UI/UX (End User)

Login

Landing page

User Home page

User Transfer page

Paytm Project 1 of 20

UI/UX (Merchant)

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

ove merchant altogether)

3. Number of services we need (merge bank server, do withdrawals directly
   - Paytm Project 1 of 20 suming 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
```

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

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.json",
  "compilerOptions": {
    "outDir": "dist"
  },
  "include": ["src"],
  "exclude": ["node_modules", "dist"]
}
```

1. Init prisma

```
npx prisma init
```

2. Update .env with the new database URL

3. Add a basic schema


```
  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.json",
   "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": [],


  "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.ReactNode}) => {
   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>
```

```
</div>
```

Paytm Project 1 of 20

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

Paytm Project 1 of 20

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

        if (existingUser) {
           const passwordValidation = await bcrypt.compare(credentials.passw
           if (passwordValidation) {
              return {
                 email: existinguser.number
```

```
            }
        }

        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`

```
const handler = NextAuth(authOptions)

p Paytm Project 1 of 20 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=
```

- Ensure u see a signin page at http://localhost:3001/api/auth/signin

e 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.ReactNode}) => {
   return <RecoilRoot>
      <SessionProvider>
         {children}
      </SessionProvider>
   </RecoilRoot>
}
```

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

Server side

```
import { getServerSession } from "next-auth"
import Paytm Project 1 of 20 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 {

  onClick: () => void;
```

```
}

p Paytm Project 1 of 20 onClick, children }: ButtonProps) => {
 return (
  <button onClick={onClick} type="button" className="text-white bg-gray-8
   {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-center">
            PayTM
        </div>
        <div className="flex flex-col justify-center pt-2">
            <Button onClick={user ? onSignout : onSignin}>{user ? "Logout" : "Login"}<
        </div>
    </div>
}
```

Paytm Project 1 of 20

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"
}
```

- PayTM should redirect the user to

```
https://bank-api-frontend.com/pay?token={token_from_step_1}
```

- 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
npm i esbuild express @types/express
```

- Update tsconfig


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

- 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
})
```

- Update DB Schema

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

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

model User {
                                                toincrement())

  name          String?
```

```
  number          String          @unique
  c
  or  Paytm Project 1 of 20  nRampTransaction[]
  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


```
cd to the right folder (packages/db)
npx prisma migrate dev --name add_balance
```

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

```
"@repo/db": "*"
```

- 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 sent b
    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: {
                                                    n your DB
                                                    ntInformation.amount)
```

```
            }
        }
    ]);

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

})

app.listen(3003);
```


Create generic appbar

nt in apps/user-app/AppbarClient.tsx

```
"use client"
import { Paytm Project 1 of 20 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>
  );
}
```

- 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 (
```

```
        <AppbarClient />
                                    {inter.className}>{children}</body>
    </html>
  );
}
```


Create sidebar

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

```
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

```
                                    ients/SidebarItem";
```

```
export default function Layout({
  h
  {
  children: React.ReactNode;
}): JSX.Element {
  return (
    <div className="flex">
      <div className="w-72 border-r border-slate-300 min-h-screen mr-4 pt-
        <div>
          <SidebarItem href={"/dashboard"} icon={<HomeIcon />} title="Home"
          <SidebarItem href={"/transfer"} icon={<TransferIcon />} title="Transfer
          <SidebarItem href={"/transactions"} icon={<TransactionsIcon />} title=
        </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
    <path stroke-linecap="round" stroke-linejoin="round" d="m2.25 12 8.954-8.95
  </svg>
}
function TransferIcon() {
    return <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24
    <path stroke-linecap="round" stroke-linejoin="round" d="M7.5 21 3 16.5m0 0L7
  </svg>
}

function TransactionsIcon() {
    return <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24
    <path stroke-linecap="round" stroke-linejoin="round" d="M12 6v6h4.5m4.5 0a
  </svg>

}
```

- Create SidebarItem component

```
                                    "next/navigation";
```

```
import React from "react";

p Paytm Project 1 of 20 n = ({ href, title, icon }: { href: string; title: string; icon: Re
const router = useRouter();
const pathname = usePathname()
const selected = pathname === href

return <div className={`flex ${selected ? "text-[#6a51a6]" : "text-slate-500"
  router.push(href);
}}>
  <div className="pr-2">
    {icon}
  </div>
  <div className={`font-bold ${selected ? "text-[#6a51a6]" : "text-slate-50
    {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";


  title,
```

```
  children,
  {
  t
 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>
 );
}
```


- Add a `Center` component that centralizes (both verticaly 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>
}
```

- 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;



}) => {
```

```
    return <select onChange={(e) => {
        alue)
    }} ay-50 border border-gray-300 text-gray-900 text-sm
      {options.map(option => <option value={option.key}>{option.value}</optic
    </select>
}
```


- Add TextInput component to `packages/ui`

```
"use client"

export const TextInput = ({
    placeholder,
    onChange,
    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">{lab
        <input onChange={(e) => onChange(e.target.value)} type="text" id="first_
    </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"


    redirectUrl: "https://www.axisbank.com/"
```

```
}];

p Paytm Project 1 of 20 = () => {
const [redirectUrl, setRedirectUrl] = useState(SUPPORTED_BANKS[0]?.redirect
return <Card title="Add Money">
<div className="w-full">
  <TextInput label={"Amount"} placeholder={"Amount"} onChange={() => {

  }} />
  <div className="py-4 text-left">
    Bank
  </div>
  <Select onSelect={(value) => {
    setRedirectUrl(SUPPORTED_BANKS.find(x => x.name === value)?.redirect
  }} options={SUPPORTED_BANKS.map(x => ({
    key: x.name,
    value: x.name
  }))} />
  <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

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

export const BalanceCard = ({amount, locked}: {
  amount: number;
  locked: number;
```

```
        <div className="flex justify-between border-b border-slate-300 pb-2">
          <div>
            Paytm Project 1 of 20
          </div>
          <div>
            {amount / 100} INR
          </div>
        </div>
        <div className="flex justify-between border-b border-slate-300 py-2">
          <div>
            Total Locked Balance
          </div>
          <div>
            {locked / 100} INR
          </div>
        </div>
        <div className="flex justify-between border-b border-slate-300 py-2">
          <div>
            Total Balance
          </div>
          <div>
            {(locked + amount) / 100} INR
          </div>
        </div>
      </Card>
    }
```

- Create user-app/components/OnRampTransaction.tsx

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

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) {
                                  Recent Transactions">
                                  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>
  }
```

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

```
import prisma from "@repo/db/client";
import { AddMoney } from "../../../components/AddMoneyCard";
import { BalanceCard } from "../../../components/BalanceCard";
import { OnRampTransactions } from "../../../components/OnRampTransaction
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>
    <div>
      <AddMoney />
    </div>
    <div>
      <BalanceCard amount={balance.amount} locked={balance.locked} />
      <OnRampTransactions transactions={transactions} />
    </div>
  </div>
}
```

```
    return {
        amount || 0,
        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.locked} /
                <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,
          token: "122",
          provider: "HDFC Bank",
        },
      },
    },
  })
  const bob = await prisma.user.upsert({
    where: { number: '9999999998' },
    update: {},
    create: {
      number: '9999999998',
      password: 'bob',
```

```
        create: {
                                    e(),
          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"
}
```

- Run command to seed db

```
npx prisma db seed
```

- Explore db

```
npx prisma studio
```

Paytm Project 1 of 20

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')
  }

}
```

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