

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

Feature planning

User login

Merchant login

UI/UX (End User)

Login

Landing Page

User Home Page

User Transfer Page

UI/UX (Merchant)

Architecture

Hot paths

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

Stack

Bootstrap the app

Adding prisma

Add a recoil/store module

Import recoil in the next.js apps

Add next-auth

Database

User-app

Merchant-app

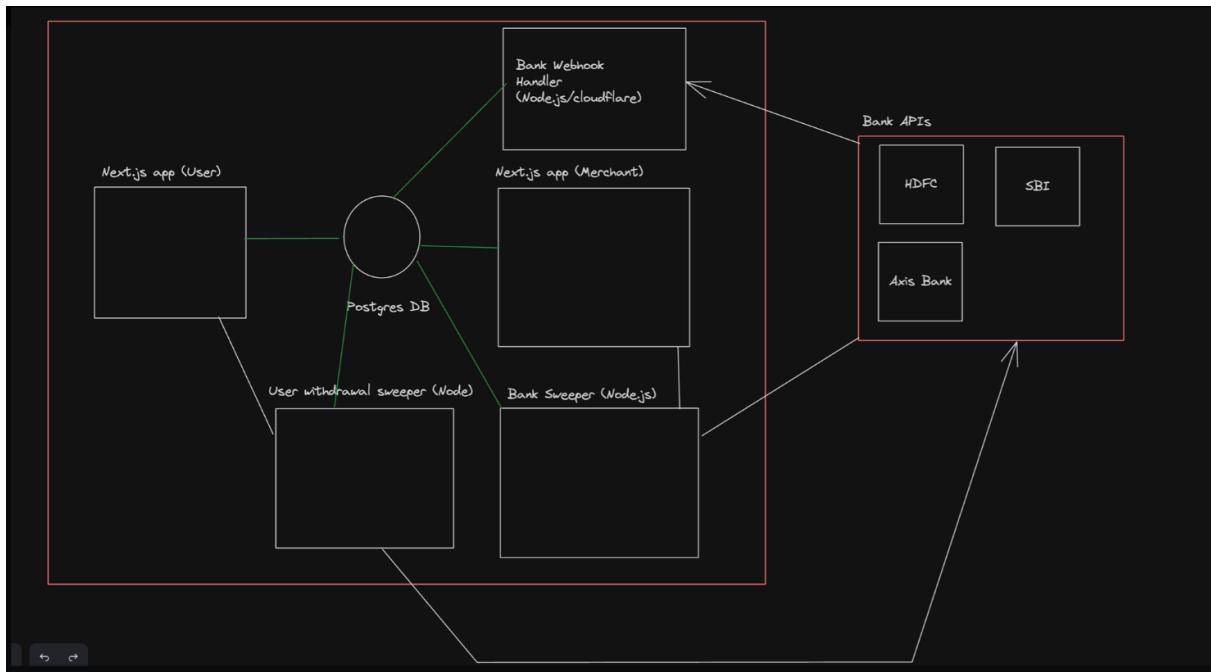
Add auth

Client side

Server side

Add an AppBar component

Checkpoint



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

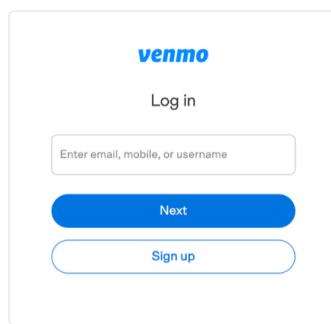
1. Auth (In this case, probably email/phone)
2. On ramp from bank, off ramp to bank
3. Support transfers via phone number/name
4. Support scanning a QR code for transferring to merchants

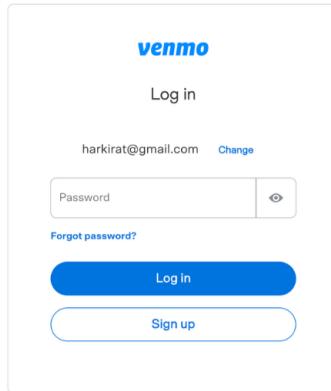
Merchant login

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

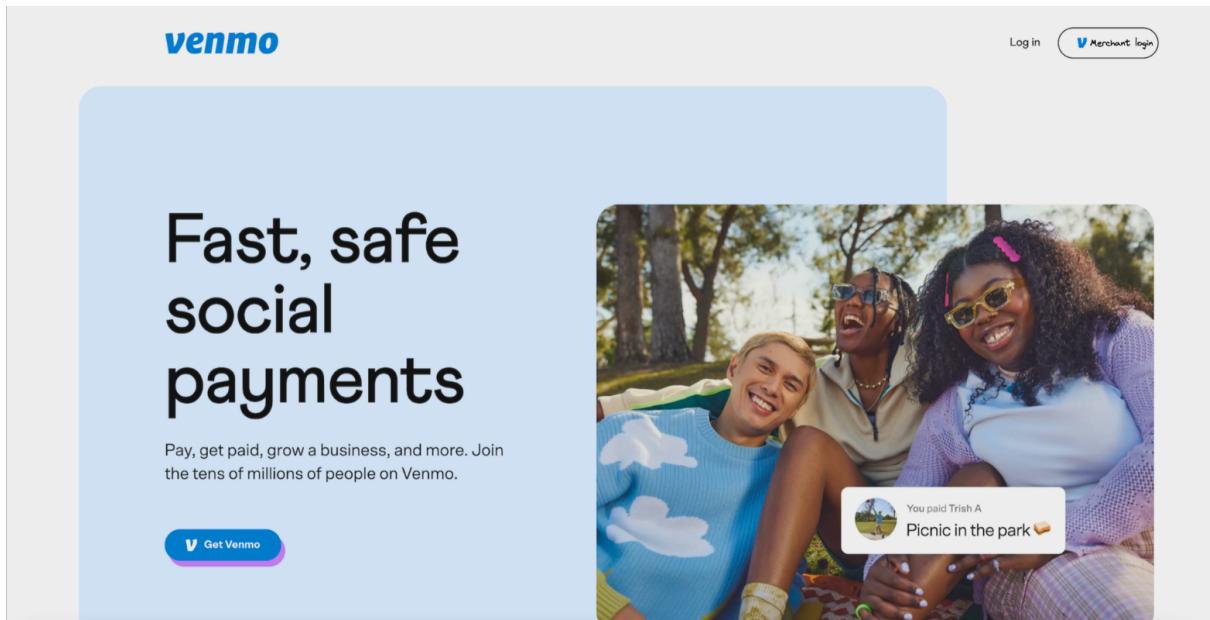
UI/UX (End User)

Login





Landing Page

The landing page for Venmo. At the top left is the "venmo" logo. On the right are links for "Log in" and "Merchant login". The main headline reads "Fast, safe social payments". Below the headline is a subtext: "Pay, get paid, grow a business, and more. Join the tens of millions of people on Venmo." At the bottom left is a blue "Get Venmo" button. To the right is a photograph of three young people smiling outdoors. A white callout box on the right side of the photo contains the text: "You paid Trish A" and "Picnic in the park 🍔".

User Home Page

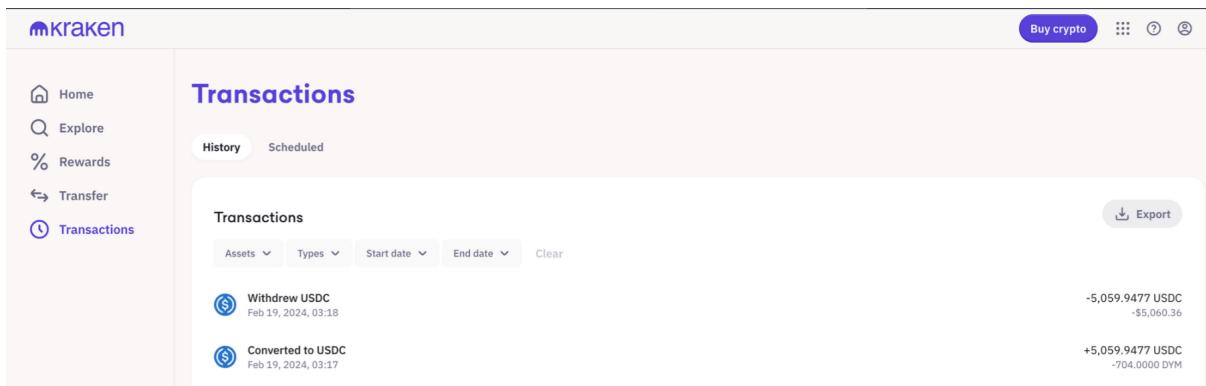
The Kraken Home page features a sidebar with links: Home, Explore, Rewards, Transfer, and Transactions. The main area displays a portfolio value of \$0.00 with a chart from February 20 to March 23. It includes buttons for Buy, Sell, Convert, Deposit, and Withdraw. A sidebar on the right offers a 'Set up recurring buys' feature with a 'Get started' button.

User Transfer Page

The Transfer page shows payment options like UPI Options, Credit/Debit/ATM Card, Book Now Pay Later, Net Banking, Gift Cards & e-wallets, EMI, and GooglePay. It also displays INR Balance, Funding limits, and Recent transactions.

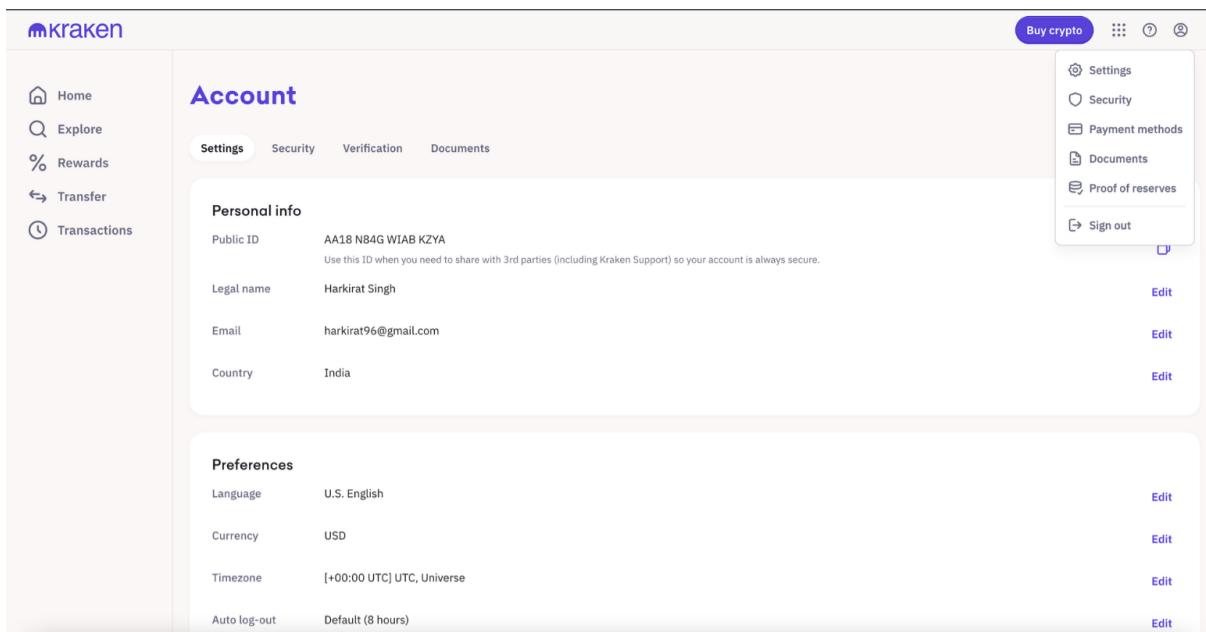
Total balance	0 BTC
Order balance	0 BTC
Staking balance	0 BTC
Available balance	0 BTC

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



The screenshot shows the Kraken Transactions page. The left sidebar includes links for Home, Explore, Rewards, Transfer, and Transactions. The main content area is titled "Transactions" and has tabs for "History" and "Scheduled". A search bar at the top of the list allows filtering by Assets, Types, Start date, End date, and Clear. Below the search bar is a table of transactions:

Type	Date	Description	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



The screenshot shows the Kraken Account page. The left sidebar includes links for Home, Explore, Rewards, Transfer, and Transactions. The main content area is titled "Account" and has tabs for "Settings", "Security", "Verification", and "Documents". A sidebar on the right provides quick access to "Settings", "Security", "Payment methods", "Documents", "Proof of reserves", and "Sign out". The "Settings" section contains two sections: "Personal info" and "Preferences".

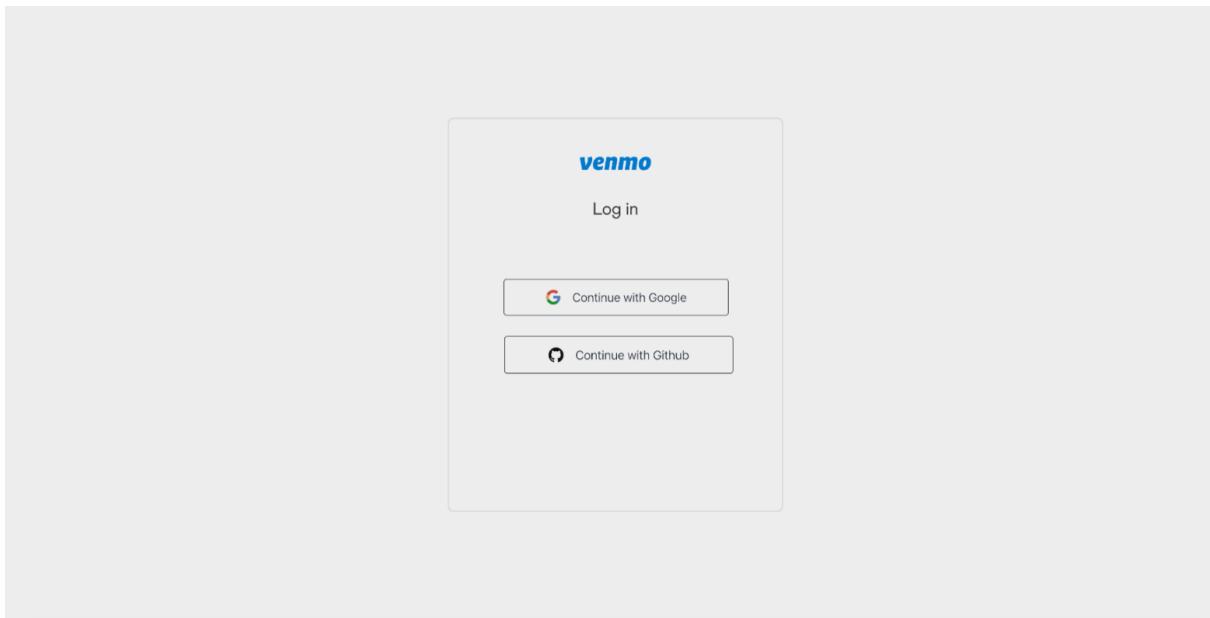
Personal info

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

Preferences

Language	U.S. English
Currency	USD
Timezone	[+00:00 UTC] UTC, Universe
Auto log-out	Default (8 hours)

UI/UX (Merchant)

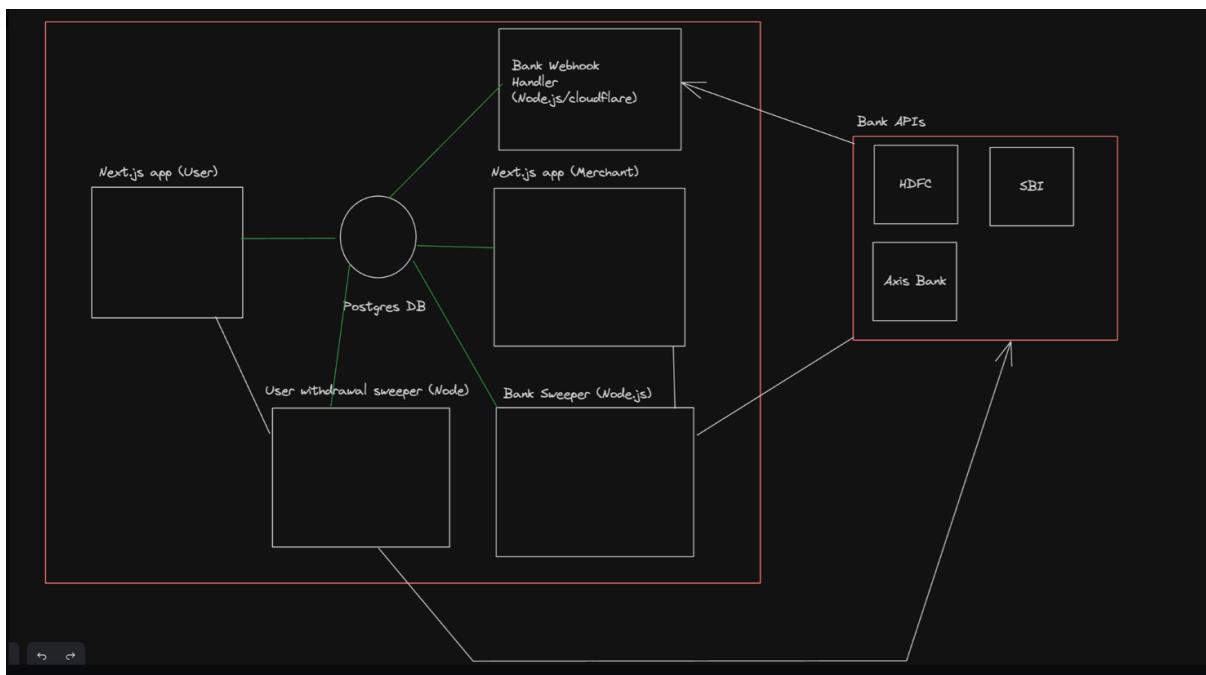


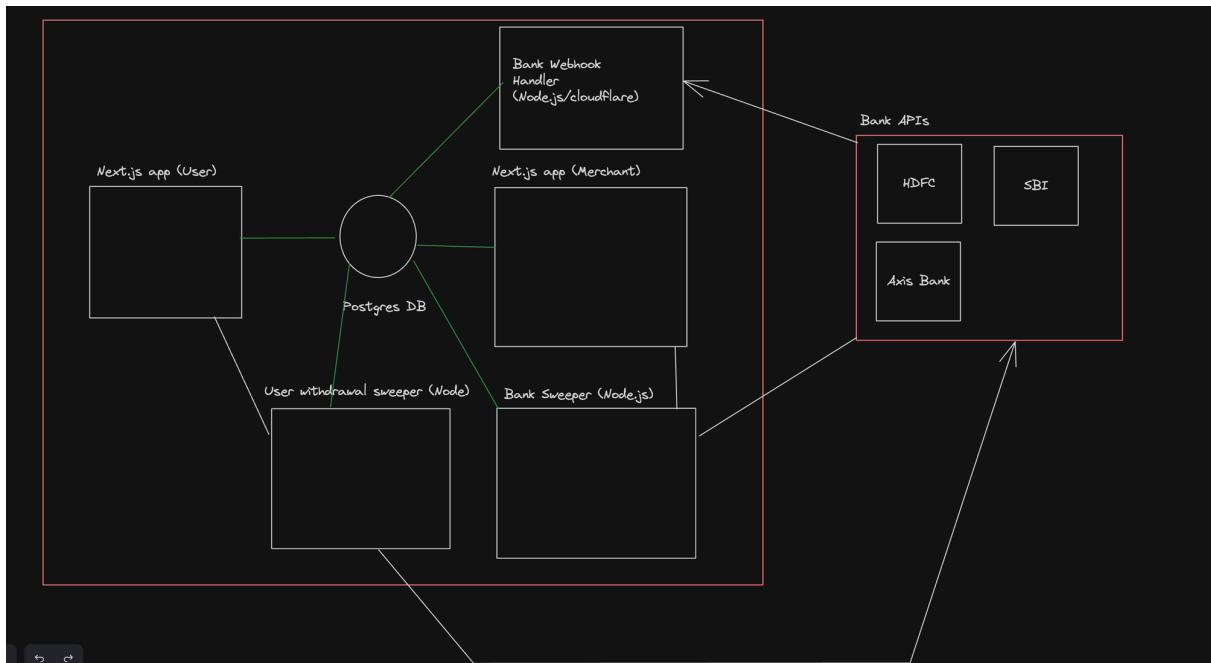
A screenshot of the Kraken home page. The header includes the Kraken logo, a "Buy crypto" button, and a menu icon. The main area has a light gray background. On the left, there's a sidebar with "Home" selected and a "Transactions" option. The center features a large purple banner with the text "Good afternoon, Harkirat". Below it, the portfolio value is shown as "\$0.00". A timeline at the bottom shows dates from 20 FEB to 23 MAR, with a "1M" button highlighted. At the bottom are five circular buttons labeled "Buy", "Sell", "Convert", "Deposit", and "Withdraw". To the right, there's a section titled "Set up recurring buys" with a "Get started" button. A small illustration of a purple battery with yellow dots is also present.

The screenshot shows the Kraken 'Transactions' page. The left sidebar includes links for Home, Explore, Rewards, Transfer, and Transactions. The main area is titled 'Transactions' with tabs for History and Scheduled. A search bar allows filtering by Assets, Types, Start date, End date, and Clear. Two transactions are listed:

- 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)

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 - Auxiliary 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.json",  
  "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.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": [],
  "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.ReactNode}) => {
    return <RecoilRoot>
        {children}
    </RecoilRoot>
}
```

- Update `layout.tsx`

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

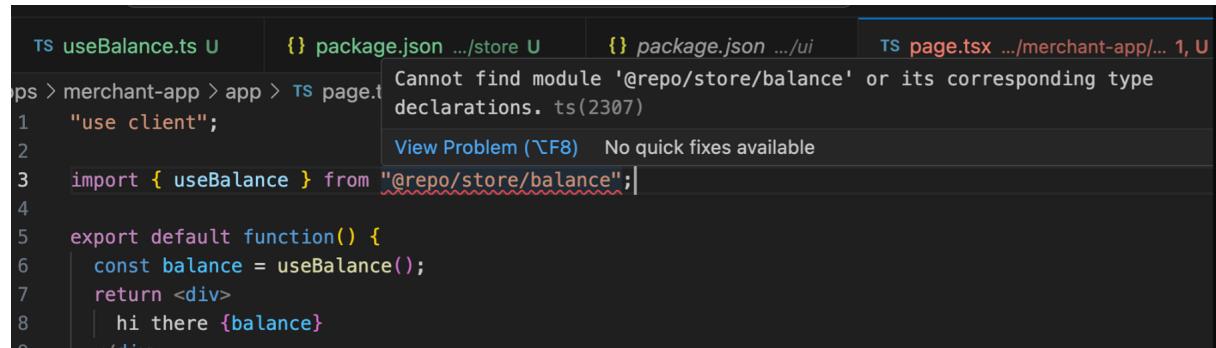
- Create a simple client component and try using the `useBalance` hook in there

```
"use client";

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

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

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



The screenshot shows a code editor with several tabs open. The active tab is 'page.tsx'. The code contains a import statement from '@repo/store/balance'. A tooltip or status bar message indicates: 'Cannot find module '@repo/store/balance' or its corresponding type declarations. ts(2307)'. There is also a link 'View Problem (F8)' and a note 'No quick fixes available'.

```
1 "use client";
2
3 import { useBalance } from "@repo/store/balance";
4
5 export default function() {
6   const balance = useBalance();
7   return <div>
8     | hi there {balance}
9   </div>
```

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/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-auth
      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.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"

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.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
    })
}

```

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-gray-800 hover:bg-gray-900 focus:outline-none focus:ring-4 focus:ring-gray-300 font-medium rounded-lg text-sm px-5 py-2.5 me-2 mb-2">
      {children}
    </button>
  );
};
```

- Create a `ui/Appbar` component that is highly generic (doesn't 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"}
            </Button>
        </div>
    </div>
}

```

Checkpoint

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