Week 17.2

Building Paytm (2/3)

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 (2/3)

Checkpoint
On Ramping
    Creating a dummy bank server
    Creating a bank_webhook_handler Node.js project

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
npx tsc --init
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 {
  id                Int                 @id @default(autoin
crement())
  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], refere
nces: [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)
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 by them
```

```
    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.userI
d)
                },
                data: {
                    amount: {
                        // You can also get this from your
DB
                        increment: Number(paymentInformatio
n.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);
```

💡 Can you use `.update` over here? Why did I have to use `.updateMany` considering tokens on onRampTransaction and userId on balance are unique