Recap Everything, Build PayTM

(https://daily-code-web.vercel.app/tracks/oAjvkeRNZThPMxZf4aX5/JLaLbhDuYn3h5Cn7WJu1)

Step 1: What are we building , Cloning the starter repository

We're building a PayTM like application that let's users send money to each other given an initial dummy balance.

Explore the Repository

The repo is basic express+ react + tailwind boilerplate

Backend

1. Express - HTTP Server
2. mongoose - ODM to connect to MongoDB
3. zod - input validation

index.js

```
const express = require("express");
const app = express();
```

Frontend

1. React - Frontend framework
2. Tailwind - Styling Framework

App.jsx

```
function App() {
    return (
      <div>
          Hello world
      </div>
    )
  }
  export default App
```

Step 2: User Mongoose schemas

We need to support 3 routes for user authentication

1. Allow user to sign up.
2. Allower user to sign in.
3. Allow user to update their information (firstName, lastName, password)

To start off, create the mongo schema for the users table

1. Create a new file (db.js) in the root folder
2. Import mongoose and connect to a database of your choice
3. Create the mongoose schema for the users table
4. Export the mongoose model from the file (call it User)

db.js

```
const mongoose = require('mongoose')
mongoose.connect('mongodb+srv://admin:KmCihXn0l1podXRj@cluster0.9gr3ic2.mo
ngodb.net/')

// create a schema for Users table
const UserSchema = new mongoose.Schema({
    username: String,
    fastName: String,
```

```
    lastName: String,
    password: String
})

// Create a model from the schema
const User = mongoose.model('User', UserSchema);
module.exports= {
    User
};
```

Elegant Solution:

```
// backend/db.js
const mongoose = require('mongoose');

// Create a Schema for Users
const userSchema = new mongoose.Schema({
    username: {
        type: String,
        required: true,
        unique: true,
        trim: true,
        lowercase: true,
        minLength: 3,
        maxLength: 30
    },
    password: {
        type: String,
        required: true,
        minLength: 6
    },
    firstName: {
        type: String,
        required: true,
        trim: true,
        maxLength: 50
    },
    lastName: {
        type: String,
        required: true,
```

```
        trim: true,
        maxLength: 50
    }
});

// Create a model from the schema
const User = mongoose.model('User', userSchema);

module.exports = {
    User
};
```

Step 3: Creating routing file structure

In the index.js file, route all the request to /api/v1 to a apiRouter defined in backend/routes/index.js

Step 1:

Create a new file **backend/routes/index.js** that exports a new express router.

(How to create a router - https://www.geeksforgeeks.org/express-js-express-router-function/)

```
const express = require('express');
const router = express.router();

module.exports = router;
```

Step 2:

Import the router in index.js and route all requests from **/api/v1** to it

backend/index.js

```
const express = require("express");
const app = express();

const rootRouter = require("./routes/index");
```

```
app.use("/api/v1",rootRouter)
```

Step 4: Route user requests

1. Create a new user router

Define a new router in **backend/routes/user.js** and import it in the index router. Route all requests that go to **/api/v1/user** to the user router.

backend/routes/user.js

```
const express = require('express')
const router = express.router();

module.express = router;
```

2. Create a new user router

Import the useRouter in **backend/routes/index.js** so all request to **/api/v1/user** get routed to the userRouter.

```
const express = require('express');
const userRouter = require('./user');

const router = express.Router();
router.use("/user",userRouter)

module.exports = router;
```

Step 5: Add cors, body parser and jsonwebtoken

1. Add cors

Since our frontend and backend will be hosted on separate routes, add the cors middleware to **backend/index.js**

npm install cors

```
const express = require("express");
const cors = require('cors')

const app = express();
app.use(cors())

const rootRouter = require("./routes/index");
app.use("/api/v1",rootRouter)
```

2. Add body-parser

Since we have to support the JSON body in post requests, add the express body parser middleware to **backend/index.js**

You can use the body-parser npm library, or use express.json

npm install body-parser

```
const express = require("express");
const cors = require('cors')
const rootRouter = require("./routes/index");
// const bodyParser = require('body-parser')

const app = express();
app.use(cors())
// no need to use this anymore
// app.use(bodyParser.json()); //utilizes the body-parser package
app.use(express.json())

app.use("/api/v1",rootRouter)
```

3. Add jsonwebtoken

We will be adding authentication soon to our application, so install jsonwebtoken library. It'll be useful in the next slide

npm install jsonwebtoken

4. Export JWT_SECRET

Export a JWT_SECRET from a new file **backend/config.js**

```
module.exports = {
    JWT_SECRET:"password"
    // JWT SECRET
}
```

5. Listen on port **3000**

Make the express app listen on PORT 3000 of your machine.

backend/index.js

```
const express = require("express");
const cors = require('cors')
const rootRouter = require("./routes/index");
// const bodyParser = require('body-parser')

const app = express();
app.use(cors())
// no need to use this anymore
// app.use(bodyParser.json()); //utilizes the body-parser package
app.use(express.json())

app.use("/api/v1",rootRouter)
app.listen(3000);
```

Step 6: Add backend auth routes

In the user router (**backend/routes/user**), add 3 new routes.

1. **Signup**

This route needs to get user information, do input validation using zod and store the information in the database provided

1. Inputs are correct (validated via zod)
2. Database doesn't already contain another user

If all goes well, we need to return the user a jwt which has their user id encoded as follows -

{

userId: "userId of newly added user"

}

```
const express = require('express')
const router = express.Router();
const app = express();
app.use(express.json())
app.post('/api/v1/user/signup',(req,res) => {
    const uName = req.body.username;
    const fName = req.body.firstName;
    const lName = req.body.lastName;
    const password = req.body.password;

})

module.exports = router;
```

Now lets add zod logic in it

npm install zod

```
const express = require('express')
const router = express.Router();
const zod = require('zod');
const {User} = require("../db");
const jwt = require("jsonwebtoken")
const { JWT_SECRET } = require('../config');

const signupBody = zod.object({
    username: zod.string().email(),
    firstName: zod.string(),
    lastName: zod.string(),
    password: zod.string()
})
```

```
router.post('/signup',async (req,res) => {
   const { success } = signupBody.safeParse(req.body);
   if(!success){
    return res.status(411).json({
        message: "Email already taken/ Incorrect inputs"
    })
   }

   const existingUser = await User.findOne({
        username: req.body.username
   })

   if(existingUser){
    return res.status(411).json({
        message:"Email already taken/Incorrect inputs"
    })
   }

   const user = await User.create({
        username: req.body.username,
        password: req.body.password,
        firstName: req.body.firstName,
        lastName: req.body.lastName
   })
   const userId = user._id;
   const token = jwt.sign({
    userId
   }, JWT_SECRET);

   res.status(200).json({
    message: "User created succesfully",
    token: token
   })
})

module.exports = router;
```

2. Route to sign in

‘Let an existing user sign in to get back a token

Method: POST

Route: /api/v1/user/signin

Body:

```
{
    username: "name@gmail.com",
    password: "123456"
}
```

Response:

Status code :- 200

```
{
    token:"jwt"
}
```

Status code :- 411

```
{
    message: "Error while logging in"
}
```

Solution

```
const signinBody = zod.object({
    username: zod.string().email(),
    password: zod.string()
})

router.post('/signin',async(req,res)=> {
    const {success} = signinBody.safeParse(req.body);
    if(!success){
        return res.status(411).json({
            message:"Incorrect inputs"
        })
    }

    const user = await User.findOne({
        username: req.body.username,
```

```
        password: req.body.password
    })

    if(user){
        const token = jwt.sign({
            userId: user._id
        }, JWT_SECRET);
        res.status(200).json({
            token:token
        })
        return;
    }

    res.status(411).json({
        message:"Error while logging in"
    })
})

module.exports = router;
```

Step 7 — Middleware

Now that we have a user account, we need to **gate** routes which authenticated users can hit.

For this, we need to introduce an auth middleware

Create a **middleware.js** file that exports an **authMiddleware** function

1. Checks the headers for an Authorization header (Bearer <token>)
2. Verifies that the token is valid
3. Puts the **userId** in the request object if the token checks out.
4. If not, return a 403 status back to the user.

```
const {JWT_SECRET} = require("./config")
const jwt = require("jsonwebtoken")

const authMiddleware = (req,res,next) => {
```

```
    const authHeader = req.headers.authorization;
    if(!authHeader || !authHeader.startsWith('Bearer ')){
        return res.status(403).json({});
    }
    const token = authHeader.split(' ')[1];

    try {
        const decoded = jwt.verify(token, JWT_SECRET);
        req.userId = decoded.userId;
        next();
    }catch(err){
        return res.status(403).json({});
    }
};

module.exports ={
    authMiddleware
}
```

Step 8 – User routes

1. Route to update user information

User should be allowed to **optionally** send either or all of

1. password
2. firstName
3. lastName

Whatever they send, we need to update it in the database for the user.

Use the middleware we defined in the last section to authenticate the user.

Method: PUT

Route: /api/v1/user

Body:

```
{
```

```
    password: "new_password",
    firstName: "updated_first_name",
    lastName: "updated_first_name",
}
```

Response:

Status code – 200

```
{
    message: "Updated successfully"
}
```

Status code – 411(Password too small..)

```
{
    message: "Error while updating information"
}
```

Solution:

```
const updateBody = zod.object({
    password: zod.string().optional(),
    firstName: zod.string().optional(),
    lastName: zod.string().optional()
})

router.put("/", authMiddleware, async(req,res) => {
    const {success} = updateBody.safeParse(req.body)
    if(!success){
        res.status(411).json({
            message:"Error while updating information"
        })
    }

    await User.updateOne({_id: req.userId}, req.body);
    res.status(200).json({
        message:"Updated succesfully"
    })
})
```

2. Route to get users from the backend, filterable via firstName/ lastName
This is needed so users can search for their friends and send them money

Method: GET

Route: /api/v1/user/bulk

Query Parameter: ?filter=harkirat

Response:

Status code- 200

```
{
    users: [{
        firstName: "",
        lastName: "",
        _id: "id of the user"
    }]
}
```

Solution

```
router.get("/bulk", async (req, res) => {
    const filter = req.query.filter || "";
    const users = await User.find({
        $or: [{
            firstName: {
                "$regex": filter
            }
        }, {
            lastName: {
                "$regex": filter
            }
        }]
    })
    res.json({
        user: users.map(user => ({
```

```
            username: user.username,
            firstName: user.firstName,
            lastName: user.lastName,
            _id: user._id
        }))
    })
})
```

User.js

```
const express = require('express')

const router = express.Router();
const zod = require('zod');
const {User} = require("../db");
const jwt = require("jsonwebtoken")
const { JWT_SECRET } = require('../config');
const {authMiddleware} = require('../middleware')

const signupBody = zod.object({
    username: zod.string().email(),
    firstName: zod.string(),
    lastName: zod.string(),
    password: zod.string()
})

router.post('/signup',async (req,res) => {
   const { success } = signupBody.safeParse(req.body);
   if(!success){
    return res.status(411).json({
        message: "Email already taken/ Incorrect inputs"
    })
   }

   const existingUser = await User.findOne({
        username: req.body.username
   })

   if(existingUser){
```

```
    return res.status(411).json({
        message:"Email already taken/Incorrect inputs"
    })
   }

   const user = await User.create({
        username: req.body.username,
        password: req.body.password,
        firstName: req.body.firstName,
        lastName: req.body.lastName
   })
   const userId = user._id;
   const token = jwt.sign({
    userId
   }, JWT_SECRET);

   res.status(200).json({
    message: "User created succesfully",
    token: token
   })
})

const signinBody = zod.object({
    username: zod.string().email(),
    password: zod.string()
})

router.post('/signin',async(req,res)=> {
    const {success} = signinBody.safeParse(req.body);
    if(!success){
        return res.status(411).json({
            message:"Incorrect inputs"
        })
    }

    const user = await User.findOne({
        username: req.body.username,
        password: req.body.password
    })
```

```
    if(user){
        const token = jwt.sign({
            userId: user._id
        }, JWT_SECRET);
        res.status(200).json({
            token:token
        })
        return;
    }

    res.status(411).json({
        message:"Error while logging in"
    })
})

const updateBody = zod.object({
    password: zod.string().optional(),
    firstName: zod.string().optional(),
    lastName: zod.string().optional()
})

router.put("/", authMiddleware, async(req,res) => {
    const {success} = updateBody.safeParse(req.body)
    if(!success){
        res.status(411).json({
            message:"Error while updating information"
        })
    }

    await User.updateOne({_id: req.userId}, req.body);
    res.status(200).json({
        message:"Updated succesfully"
    })
})

router.get("/bulk", async (req, res) => {
    const filter = req.query.filter || "";
    const users = await User.find({
        $or: [{
            firstName: {
```

```
                "$regex": filter
            }
        }, {
            lastName: {
                "$regex": filter
            }
        }]
    })
    res.json({
        user: users.map(user => ({
            username: user.username,
            firstName: user.firstName,
            lastName: user.lastName,
            _id: user._id
        }))
    })
})

module.exports = router;
```

Step 9 — Create Bank related Schema

Update the db.js file to add one new schemas and export the respective models

Account table

The Accounts table will store the INR balances of a user.

The schema should look like something like this:-

```
{
    userId: ObjectId (or string),
    balance: float/number
}
```

In the real world, you shouldn't store `floats` for balances in the database.

You usually store an integer which represents the INR value with

decimal places (for eg, if someone has 33.33 rs in their account,

you store 3333 in the database).

There is a certain precision that you need to support (which for india is 2/4 decimal places) and this allows you to get rid of precision errors by storing integers in your DB

```
const accountSchema = new mongoose.Schema({
    userId: {
        type: mongoose.Schema.Types.ObjectId, // Reference to User model
        ref: 'User',
        required: true
    },
    balance: {
        type: Number,
        required: true
    }
});

const Account = mongoose.model('Account', accountSchema);

module.exports = {
    Account
}
```

db.js

```
const mongoose = require('mongoose')
mongoose.connect('mongodb+srv://admin:KmCihXn0l1podXRj@cluster0.9gr3ic2.mongodb.net/')

// create a schema for Users table
const userSchema = new mongoose.Schema({
    username: {
        type: String,
        required: true,
        unique: true,
        trim: true,
        lowercase: true,
```

```
        minLength: 3,
        maxLength: 30
    },
    password: {
        type: String,
        required: true,
        minLength: 6
    },
    firstName: {
        type: String,
        required: true,
        trim: true,
        maxLength: 50
    },
    lastName: {
        type: String,
        required: true,
        trim: true,
        maxLength: 50
    }
});

// Create a model from the schema
const accountSchema = new mongoose.Schema({
    userId: {
        type: mongoose.Schema.Types.ObjectId,
        ref:'User',
        required: true
    },
    balance:{
        type:Number,
        required: true
    }
});
const Account = mongoose.model('Account',accountSchema);
const User = mongoose.model('User', UserSchema);
module.exports= {
    User,
    Account
};
```

Step 10 — Transaction in databases

A lot of times , you want multiple database transactions to be atomic.

Either all of them should update, or none should

This is super important in the case of a bank.

Can you guess what is wrong in this code

```
const mongoose = require('mongoose')
const Account = require('./path-to-your-account-model');
const transferFunds = async(fromAccountId, toAccountId, amount) => {
    // Decrease the balance of the fromAccount
    await Account.findByIdAndUpdate(fromAccountId,{$inc:{balance:
-amount}});

    // Increase the balance of the toAccount
    await Account.findByIdAndUpdate(toAccountId,{ $inc: { balance:
amount}})
}
// Example usage
transferFunds('fromAccountId','toAccountId',100);
```

Solution:

1. What if the database crashes right after the first request (only the balance is decreased for one user, and not for the second user)
2. What if the Node.js crashesh after the first update

It would lead to a database inconsistency. Amount would get debited from the first user, and not credited into the other users account.

If a failure ever happens, the first txn should rollback\

That is what is called a transaction in a database. We need to implement a transaction on the next set of endpoints that allow users to transfer INR.

Step 11 — Initialize balances on signup

Update the **signup** endpoints to give the user a random balance between 1 and 10000. This is so we don't have to integrate with banks and give them random balances to start with.

```
// Create a new account
   await Account.create({
         userId,
         balance: 1 + Math.random()*10000
   })
```

Step 12 — Create a new router for accounts

1. **Create a new router**

All user balances should go to a different express router (that handles all requests on **/api/v1/account**)

Create a new router in **routes/account.js** and add export it.

```
const express = require('express');
const router = express.Router();
module.exports = router;
```

2. **Route requests to it**

Send all request from **/api/v1/account/*** in **routes/index.js** to the router created in step 1.

```
const express = require('express');
const userRouter = require('./user');
const accountRouter = require('./account')

const router = express.Router();

router.use("/user",userRouter);
// we are basically telling that if a request come here go here and if a
request come there go there
router.use('/account',accountRouter);
```

```
module.exports = router;
```

Step 13 – Balance and transfer Endpoints

Here, you'll be writing a bunch of API's for the core user balances. There are two endpoints that we need to implement

1. **An endpoint for user to get their balance**

Method: GET

Route: /api/v1/account/balance

Response:

Status code – 200

```
{
    balance:1000
}
```

Solution:

```
const express = require('express');
const { authMiddleware } = require('../middleware');
const {Account} = require("../db");
const router = express.Router();

router.get('/balance',authMiddleware,async(req,res)=>{
    const account = await Account.findOne({
        userId: req.userId
    })
    res.json({
        balance: account.balance
    })
})
```

```
module.exports = router;
```

2. **An endpoint for user to transfer money to another account**

METHOD: POST

Route: /api/v1/account/transfer

Body:

```
{
    to: string,
    amount: number
}
```

Responses:

Status code – 200

```
{
    message:”Transfer succesfull”
}
```

Status code – 400

```
{
    message:”Insufficient balance”
}
```

Bad solution (doesn’t use transaction)

It will work unless we want to do concurrent transactions or our database goes down in a middle of transactions.

```
const express = require('express');
const { authMiddleware } = require('../middleware');
const {Account} = require("../db");
const router = express.Router();

router.get('/balance',authMiddleware,async(req,res)=>{
    const account = await Account.findOne({
        userId: req.userId
```

```
    })
    res.json({
        balance: account.balance
    })
})

router.post("/transfer",authMiddleware,async(req,res) => {
    const {amount, to } = req.body;
    const account = await Account.findOne({
        world: req.userId
    })
    if(account.balance < amount){
        return res.status(400).json({
            message:"Insufficient balance"
        })
    }
    const toAccount = await Account.findOne({
        userId:to
    });
    if(!toAccount){
        return res.status(400).json({
            message:"Invalid account"
        })
    }
    await Account.updateOne({
        userId:req.userId
    },{
        $inc:{
            balance: -amount
        }
    })
    await Account.updateOne({
        userId:to
    },{
        $inc:{
            balance:amount
        }
    })
    res.json({
        message:"Transfer successful"
```

```
    })
})

module.exports = router;
```

Good solution in ideal situation , assuming nodejs and database both up always.

Good solution (using transaction in DB)

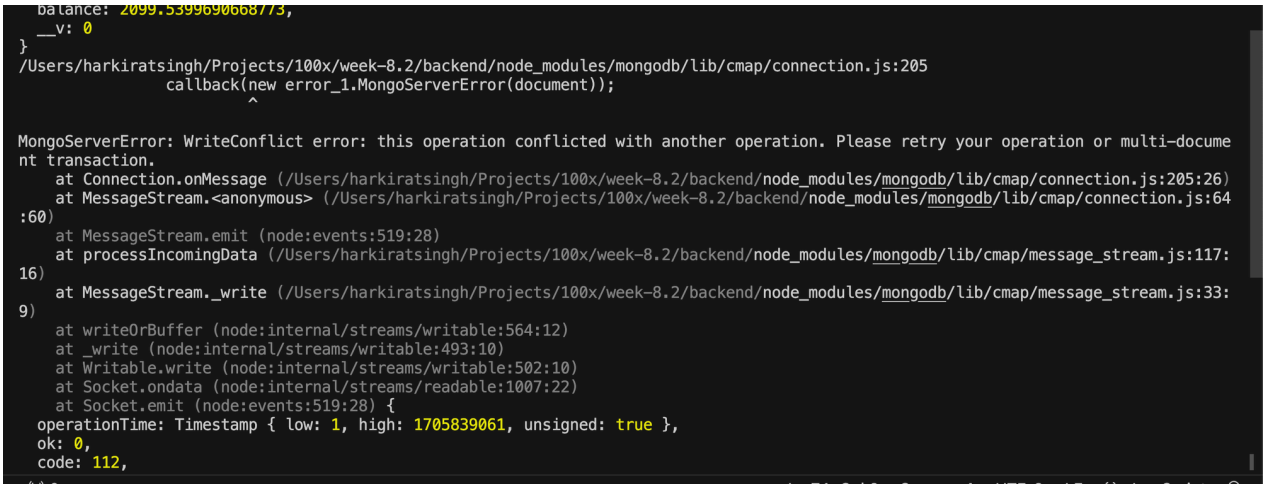
```
router.post("/transfer",authMiddleware,async(req,res)=>{
    const session = await mongoose.startSession();
    session.startTransaction();
    const{amount,to} = req.body;

    // Fetch the accounts within the transcation
    const account = await
Account.findOne({userId:req.userId}).session(session);

    if(!account || account.balance < amount){
        await session.abortTransaction();
        return res.status(400).json({
            message:'Insufficient balance'
        });
    }
    const toAccount = await Account.findOne({userId:to}).session(session);

    if(!toAccount){
        await session.abortTransaction();
        return res.status(400).json({
            message:"Invalid Account"
        })
    }

    // Perform the transfer
    await Account.updateOne({userId:req.userId},{$inc:{balance:
-amount}}).session(session);
    await
Account.updateOne({userId:to},{$inc:{balance:amount}}).session(session);
```

```
    // Commit transaction
    await session.commitTransaction();
    res.json({
        message:"Transfer succesful"
    })
})
```

Lets try to verify what will happen if we send two concurrent requests.

```
// backend/routes/account.js
const express = require('express');
const { authMiddleware } = require('../middleware');
const { Account } = require('../db');
const { default: mongoose } = require('mongoose');

const router = express.Router();

router.get("/balance", authMiddleware, async (req, res) => {
    const account = await Account.findOne({
        userId: req.userId
    });

    res.json({
        balance: account.balance
    })
});

async function transfer(req) {
    const session = await mongoose.startSession();
    session.startTransaction();
    const { amount, to } = req.body;
    // Fetch the accounts within the transaction
    const account = await Account.findOne({ userId: req.userId
}).session(session);

    if (!account || account.balance < amount) {
        await session.abortTransaction();
        console.log("Insufficient balance")
```

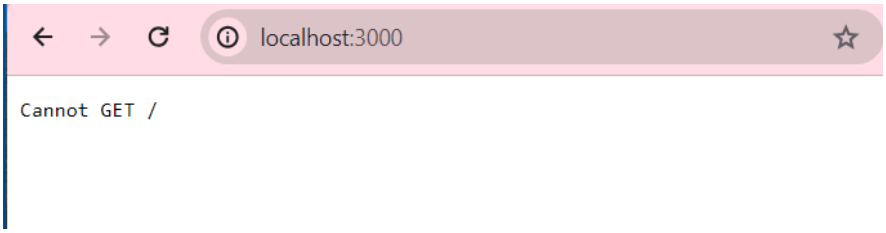
```
        return;
    }

    const toAccount = await Account.findOne({ userId: to
}).session(session);

    if (!toAccount) {
        await session.abortTransaction();
        console.log("Invalid account")
        return;
    }

    // Perform the transfer
    await Account.updateOne({ userId: req.userId }, { $inc: { balance:
-amount } }).session(session);
    await Account.updateOne({ userId: to }, { $inc: { balance: amount }
}).session(session);

    // Commit the transaction
    await session.commitTransaction();
    console.log("done")
}
// sending concurrent transaction
transfer({
    userId: "65ac44e10ab2ec750ca666a5",
    body: {
        to: "65ac44e40ab2ec750ca666aa",
        amount: 100
    }
})

transfer({
    userId: "65ac44e10ab2ec750ca666a5",
    body: {
        to: "65ac44e40ab2ec750ca666aa",
        amount: 100
    }
})
module.exports = router;
```



Lets run and check our backend

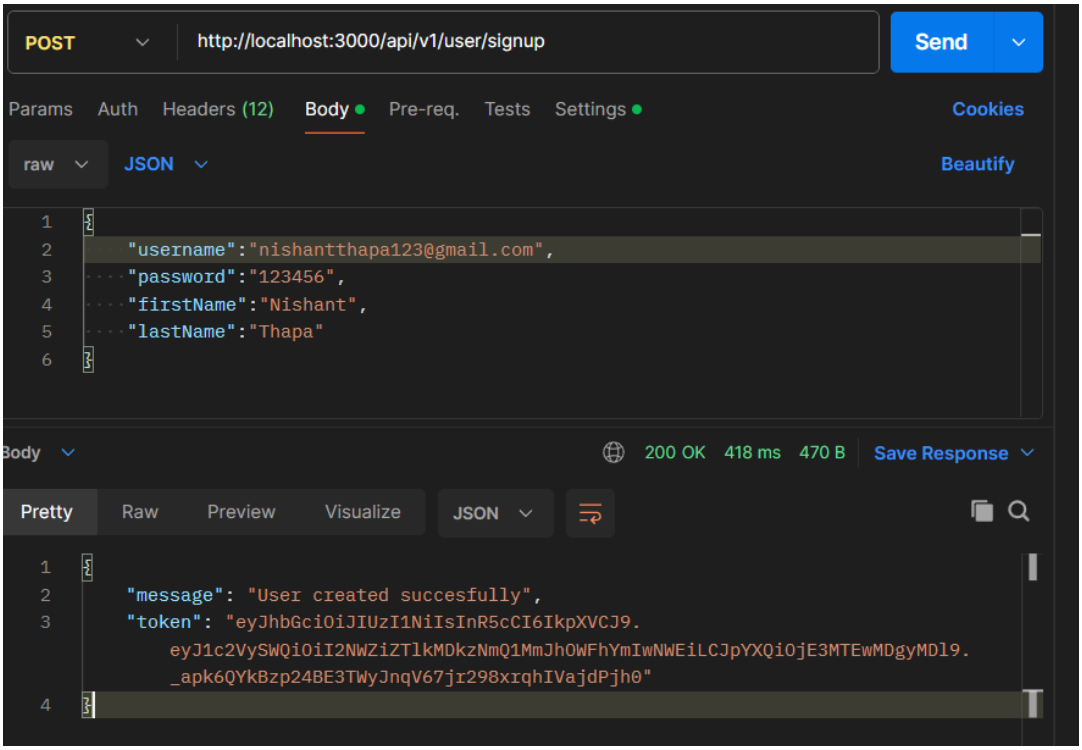
node index.js

http://localhost:3000

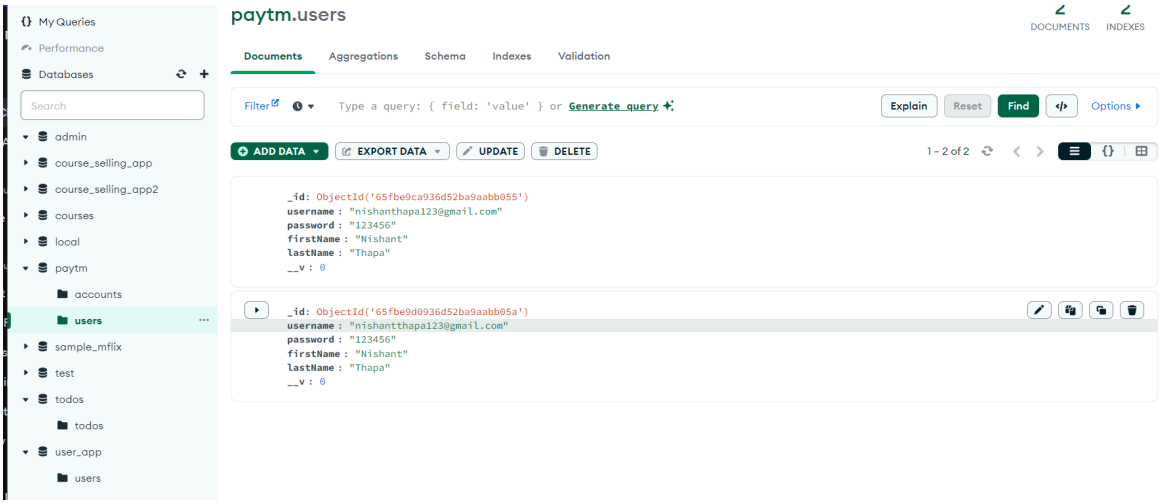


Post Request to http://localhost:3000/api/v1/user/signup endpoint

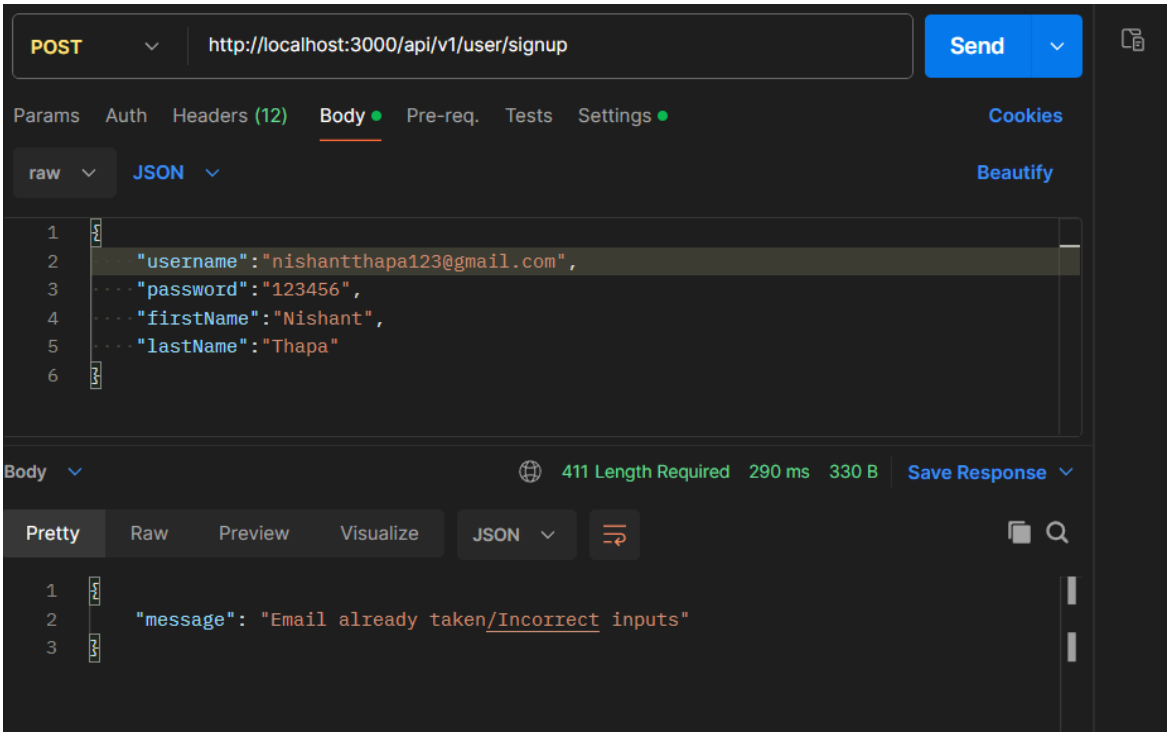
Sending a postman request



Checking our database

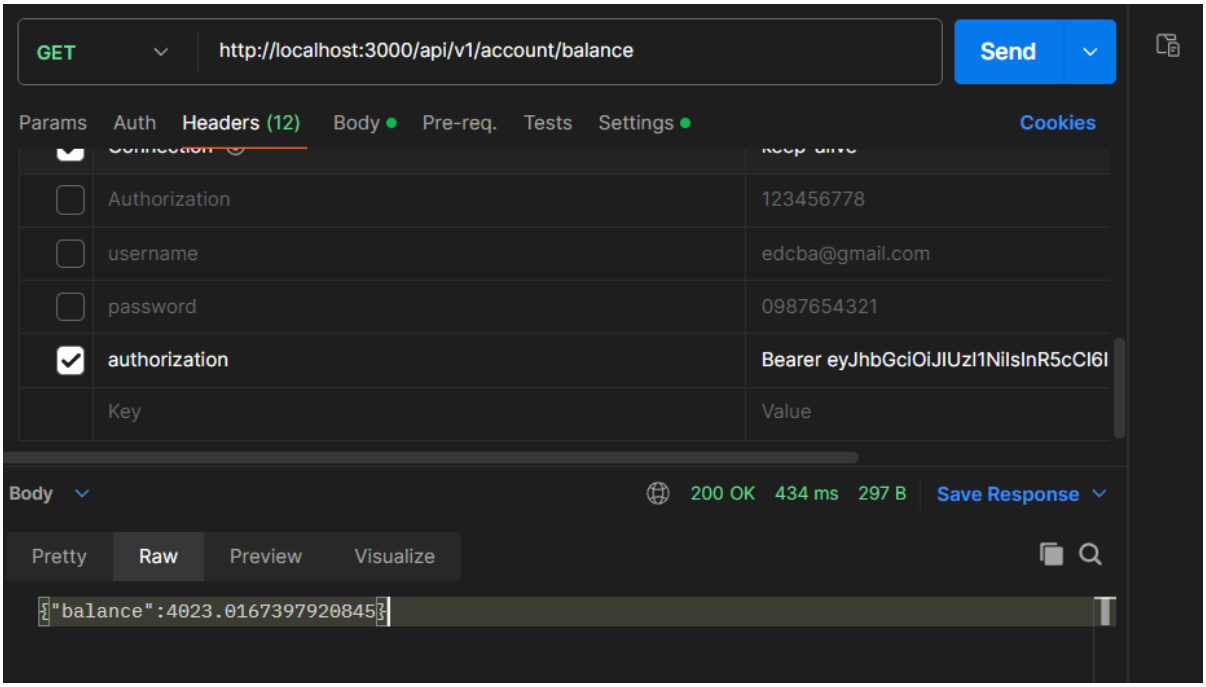


If we try to hit the endpoint with the same credentials

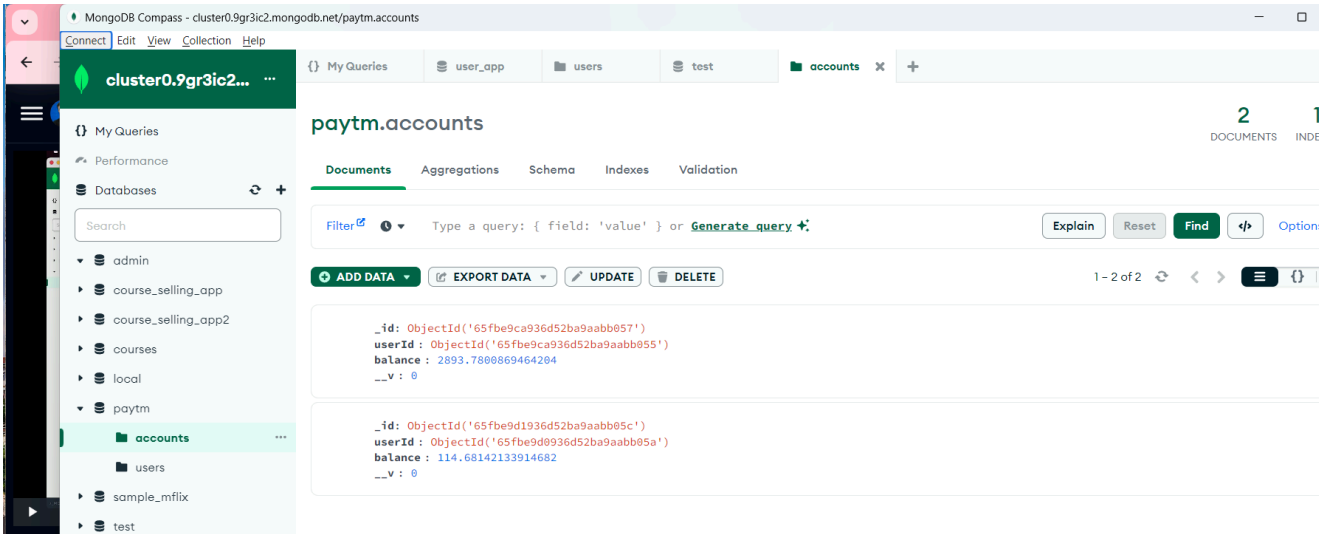


Lets check our balance

http://localhost:3000/api/v1/account/balance



We are sending the authorization token in headers which we get from the signup



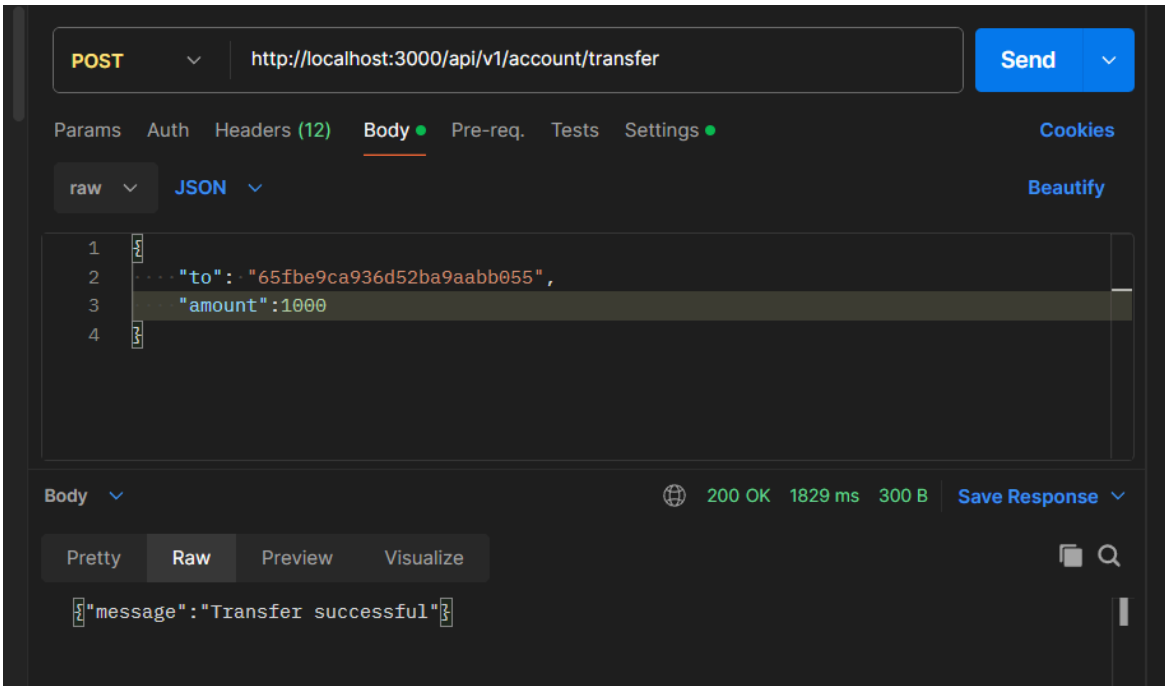
http://localhost:3000/api/v1/account/transfer

IT need to have right authorization header for the person sending the money.

Body need to have two argument

“to” to whom we are sending the money their id

“amount” how much money we are sending.



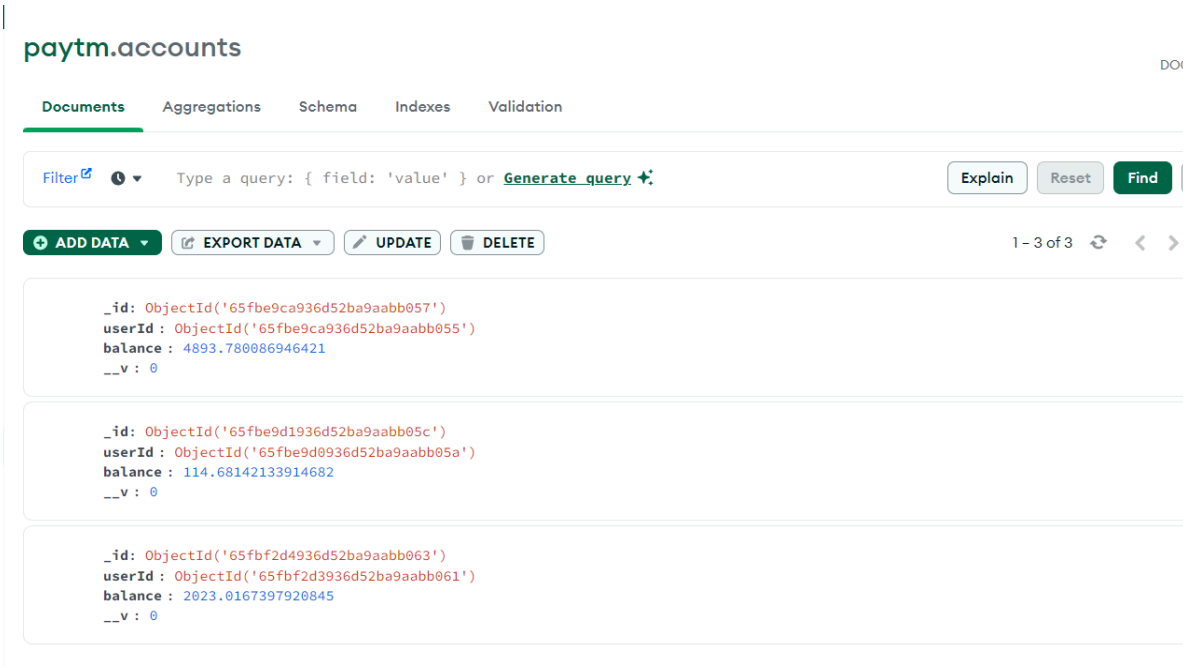
Now lets check the balance in the database.

```
_id: ObjectId('65fbe9ca936d52ba9aabb057')
userId : ObjectId('65fbe9ca936d52ba9aabb055')
balance : 3893.7800869464204
__v : 0
```

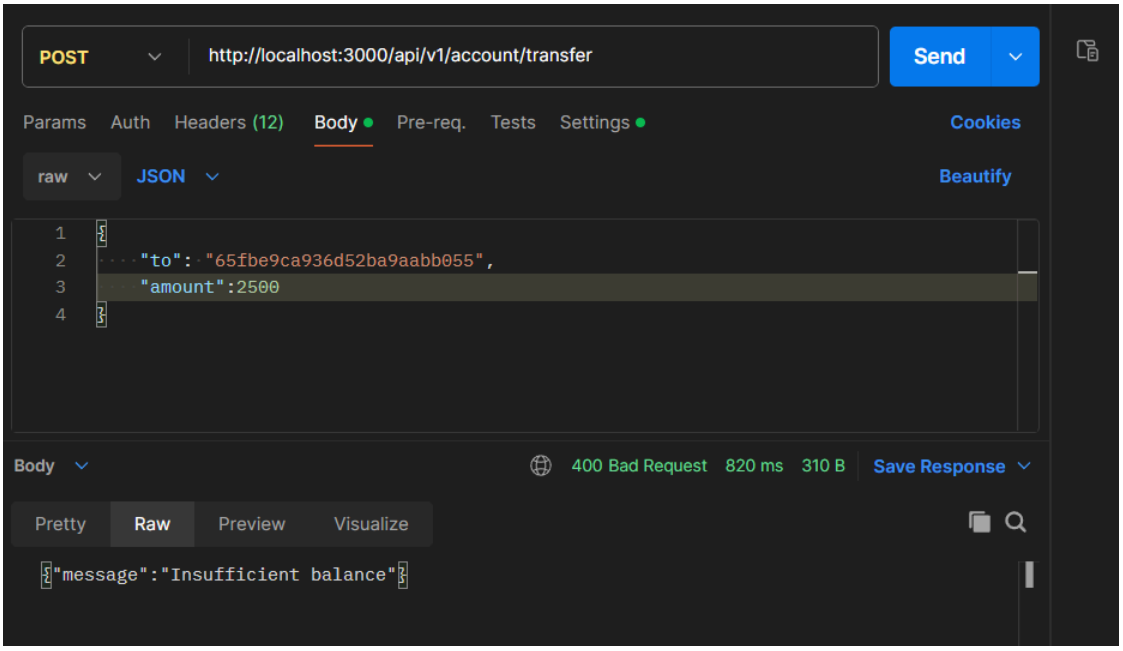
```
_id: ObjectId('65fbe9d1936d52ba9aabb05c')
userId : ObjectId('65fbe9d0936d52ba9aabb05a')
balance : 114.68142133914682
__v : 0
```

```
_id: ObjectId('65fbf2d4936d52ba9aabb063')
userId : ObjectId('65fbf2d3936d52ba9aabb061')
balance : 3023.0167397920845
__v : 0
```

We can see the balance getting decreased.



Now lets try to transfer more money then the person has



Frontend and connecting Frontend and backend

Step 15: Routing in react

Step 1 – Add routing to the react app

Import **react-router-dom** into your project and add the following routes -

1. **/signup** - The signup page
2. **/signin** - The signin page
3. **/dashboard** - Balances and see other users on the platform.
4. **/send** - Send money to other users

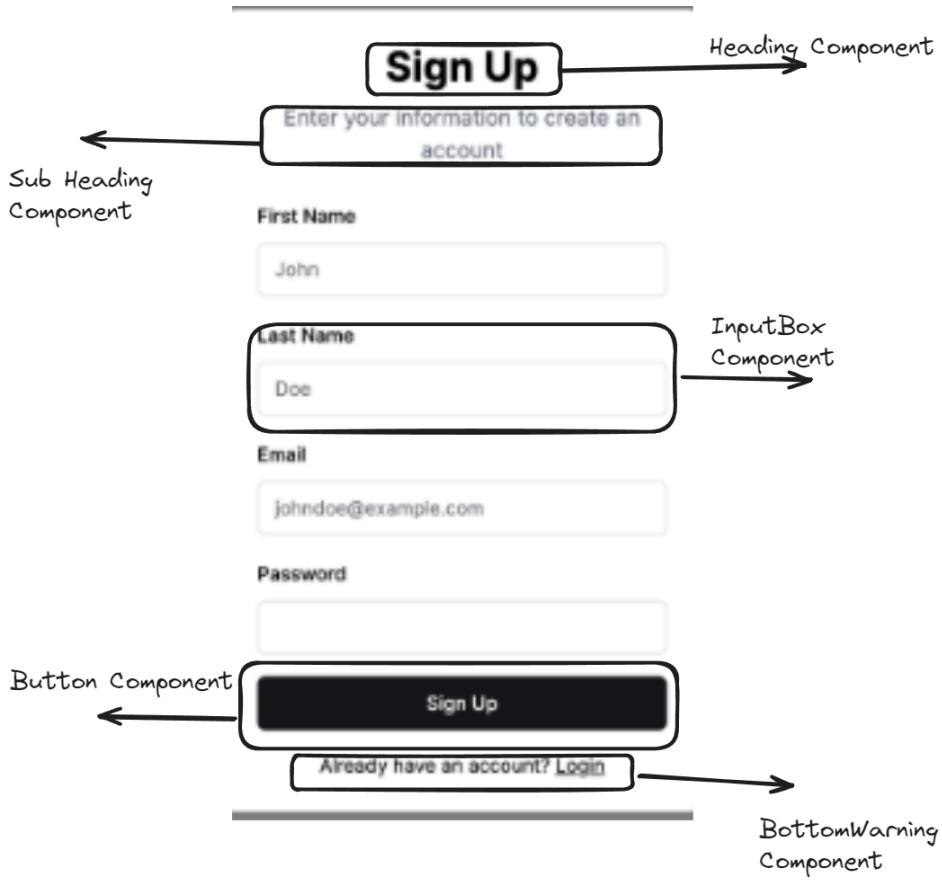
```
import { BrowserRouter, Routes, Route } from "react-router-dom";
function App() {
  return (
    <>
      <BrowserRouter>
      <Routes>
```

```
        <Route path="/signup" element={<Signup/>}></Route>
        <Route path="/signin" element={<Signin/>}></Route>
        <Route path="/dashboard" element={<Dashboard/>}></Route>
        <Route path="/send" element={<SendMoney/>}></Route>
      </Routes>
      </BrowserRouter>
    </>
  )
}
export default App
```

Step 16: Create and hookup Signup component

Whenever doing FE we break app into smaller component. Sign up can be break into smaller components

Let take example of Signup page



Example Heading component can be used in both SignIn and SignUp

Components:

- **ButtomWarning.jsx**

```
import {Link} from "react-router-dom"

export function BottomWarning({label,buttonText,to}){
    return <div className="py-2 text-sm flex justify-center">
        <div>
            {label}
        </div>
        <Link className="pointer underline pl-1 cursor-pointer" to={to}>
            {buttonText}
        </Link>
    </div>
}
```

- **Button.jsx**

```
export function Button({label, onClick}) {
    return <button onClick={onClick} type="button" class="w-full
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">{label}</button>
}
```

- **Heading.jsx**

```
export function Heading({label}) {
    return <div className="font-bold text-4xl pt-6">
      {label}
    </div>
}
```

- **InputBox.jsx**

```
export function InputBox({label, placeholder}) {
    return <div>
      <div className="text-sm font-medium text-left py-2">
        {label}
      </div>
      <input placeholder={placeholder} className="w-full px-2 py-1 border
rounded border-slate-200" />
    </div>
}
```

- **SubHeading.jsx**

```
export function SubHeading({label}) {
    return <div className="text-slate-500 text-md pt-1 px-4 pb-4">
      {label}
    </div>
  }
```

- **Balance.jsx**

```
export const Balance = ({ value }) => {
    return <div className="flex">
        <div className="font-bold text-lg">
            Your balance
        </div>
        <div className="font-semibold ml-4 text-lg">
            Rs {value}
        </div>
    </div>
}
```

- **Users.jsx**

```
import { useState } from "react"
import { Button } from "./Button"

export const Users = () => {
    // Replace with backend call
    const [users, setUsers] = useState([{
```

```
        firstName: "Harkirat",
        lastName: "Singh",
        _id: 1
    }]);

    return <>
        <div className="font-bold mt-6 text-lg">
            Users
        </div>
        <div className="my-2">
            <input type="text" placeholder="Search users..."
classNam="w-full px-2 py-1 border rounded border-slate-200"></input>
        </div>
        <div>
            {users.map(user => <User user={user} />)}
        </div>
    </>
}

function User({user}) {
    return <div className="flex justify-between">
        <div className="flex">
            <div className="rounded-full h-12 w-12 bg-slate-200 flex
justify-center mt-1 mr-2">
                <div className="flex flex-col justify-center h-full
text-xl">
                    {user.firstName[0]}
                </div>
            </div>
            <div className="flex flex-col justify-center h-ful">
                <div>
                    {user.firstName} {user.lastName}
                </div>
            </div>
        </div>

        <div className="flex flex-col justify-center h-ful">
            <Button label={"Send Money"} />
        </div>
    </div>
```

```
}
```

Pages:

- **Dashboard.jsx**

```
import { Appbar } from "../components/Appbar"
import { Balance } from "../components/Balance"
import { Users } from "../components/Users"

export const Dashboard = () => {
    return <div>
        <Appbar />
        <div className="m-8">
            <Balance value={"10,000"} />
            <Users />
        </div>
    </div>
}
```

- **SendMoney.jsx**

```
export const SendMoney = () => {
    return <div className="flex justify-center h-screen bg-gray-100">
        <div className="h-full flex flex-col justify-center">
            <div
                className="border h-min text-card-foreground max-w-md p-4
space-y-8 w-96 bg-white shadow-lg rounded-lg"
            >
                <div className="flex flex-col space-y-1.5 p-6">
                <h2 className="text-3xl font-bold text-center">Send
Money</h2>
                </div>
                <div className="p-6">
                <div className="flex items-center space-x-4">
                    <div className="w-12 h-12 rounded-full bg-green-500
flex items-center justify-center">
                    <span className="text-2xl text-white">A</span>
                    </div>
```

```
                    <h3 className="text-2xl font-semibold">Friends
Name</h3>
                </div>
                <div className="space-y-4">
                    <div className="space-y-2">
                    <label
                        className="text-sm font-medium leading-none
peer-disabled:cursor-not-allowed peer-disabled:opacity-70"
                        for="amount"
                    >
                        Amount (in Rs)
                    </label>
                    <input
                        type="number"
                        className="flex h-10 w-full rounded-md border
border-input bg-background px-3 py-2 text-sm"
                        id="amount"
                        placeholder="Enter amount"
                    />
                    </div>
                    <button className="justify-center rounded-md text-sm
font-medium ring-offset-background transition-colors h-10 px-4 py-2 w-full
bg-green-500 text-white">
                        Initiate Transfer
                    </button>
                </div>
                </div>
        </div>
      </div>
    </div>
}
```

- **Signin.jsx**

```
import { BottomWarning } from "../components/BottomWarning"
import { Button } from "../components/Button"
import { Heading } from "../components/Heading"
import { InputBox } from "../components/InputBox"
import { SubHeading } from "../components/SubHeading"
```

```
export const Signin = () => {
    return <div className="bg-slate-300 h-screen flex justify-center">
    <div className="flex flex-col justify-center">
      <div className="rounded-lg bg-white w-80 text-center p-2 h-max
px-4">
        <Heading label={"Sign in"} />
        <SubHeading label={"Enter your credentials to access your
account"} />
        <InputBox placeholder="harkirat@gmail.com" label={"Email"} />
        <InputBox placeholder="123456" label={"Password"} />
        <div className="pt-4">
          <Button label={"Sign in"} />
        </div>
        <BottomWarning label={"Don't have an account?"} buttonText={"Sign
up"} to={"/signup"} />
      </div>
    </div>
  </div>
}
```

- **Signup.jsx**

```
import {BottomWarning} from "../components/BottomWarning"
import {Button} from "../components/Button"
import {Heading} from "../components/Heading"
import {InputBox} from "../components/InputBox"
import {SubHeading} from "../components/SubHeading"

export const Signup = () => {
    return <div className="bg-slate-300 h-screen flex justify-center">
        <div className="flex flex-col justify-center">
            <div className="rounded-lg bg-white w-80 text-center p-2 h-max
px-4">
                <Heading label={"Sign up"} />
                <SubHeading label={"Enter your information to create an
account"} />
                <InputBox placeholder="JohnCena" label={"First Name"}/>
                <InputBox placeholder="YouCanSeeMe" label={"Last Name"} />
                <InputBox placeholder="nishant1234@gmail.com"
label={"Email"}/>
```

```
                <InputBox placeholder="1234" label={"password"} />
                <div className="pt-4">
                    <Button label={"Sign up"} />
                </div>
                <BottomWarning label={"Already have an account?"}
buttonText={"Sign in"} to={"/signin"}/>
            </div>

        </div>

    </div>
}
```

Right now our components don’t have any functionality they are just

Lets First add Functionality in the Signup page

Adding functionality in **InputBox.jsx** component

```
export function InputBox({label, placeholder, onChange}) {
    return <div>
      <div className="text-sm font-medium text-left py-2">
        {label}
      </div>
      <input onChange={onChange} placeholder={placeholder}
className="w-full px-2 py-1 border rounded border-slate-200" />
      {/* as the input changes iam going to call the parent onChange
fucntion */}
    </div>
}
```

Now some changes in Signup.jsx

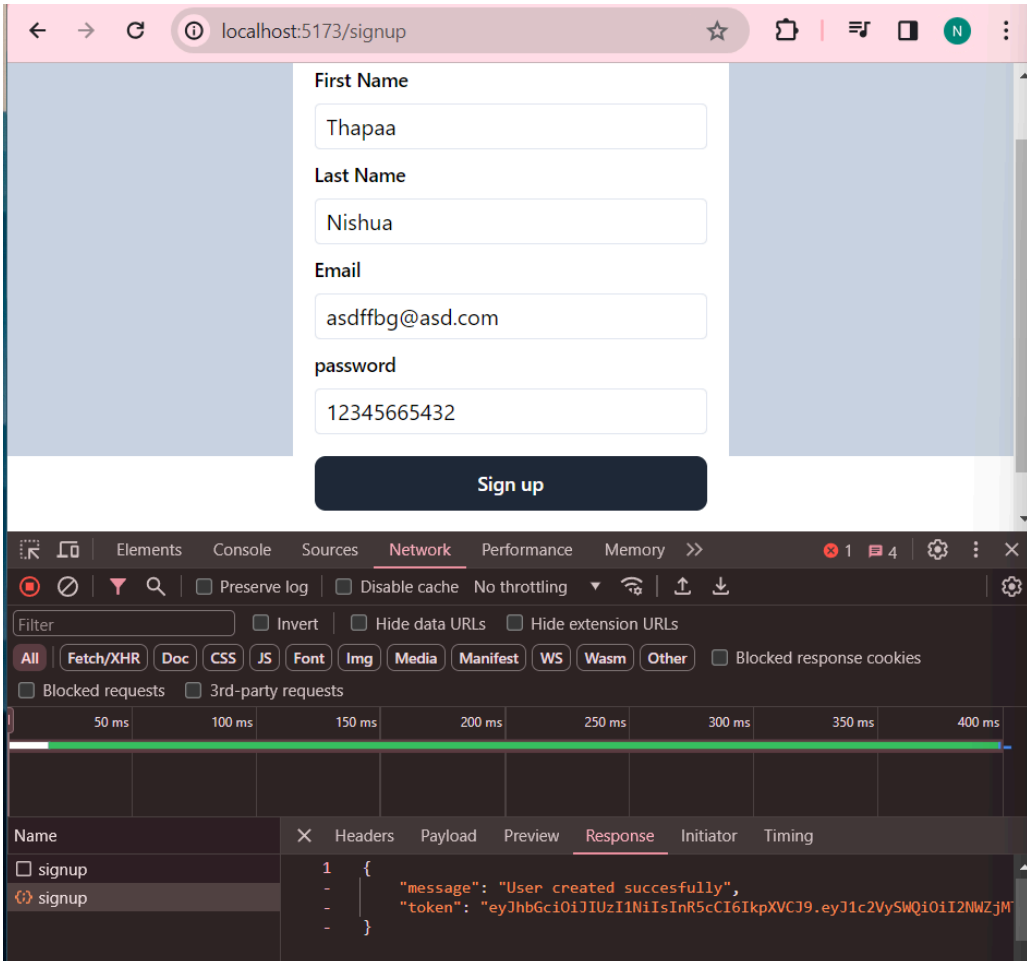
```
import { useState } from "react"
import axios from "axios"
import {BottomWarning} from "../components/BottomWarning"
import {Button} from "../components/Button"
import {Heading} from "../components/Heading"
import {InputBox} from "../components/InputBox"
```

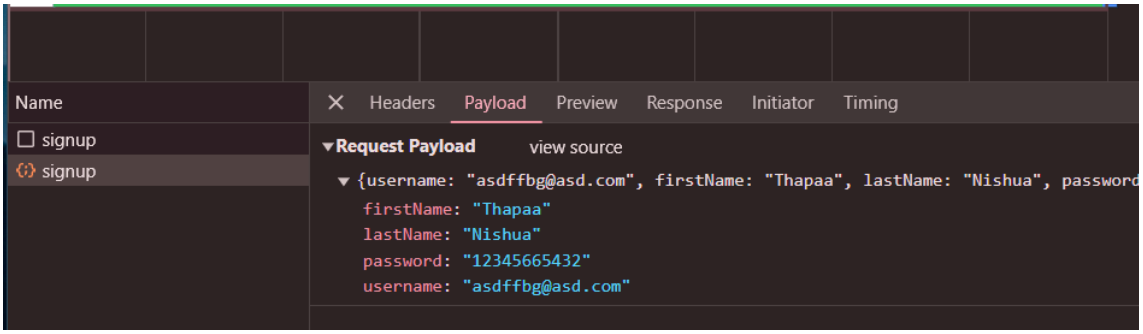
```
import {SubHeading} from "../components/SubHeading"

export const Signup = () => {
    const [firstName, setFirstName] = useState("");
    const [lastName, setLastName] = useState("");
    const [username, setUsername] = useState("");
    const [password, setPassword] = useState("");

    return <div className="bg-slate-300 h-screen flex justify-center">
        <div className="flex flex-col justify-center">
            <div className="rounded-lg bg-white w-80 text-center p-2 h-max
px-4">
                <Heading label={"Sign up"} />
                <SubHeading label={"Enter your information to create an
account"} />
                <InputBox onChange={(e) => {
                    setFirstName(e.target.value)
                    // populated the input variable
                }} placeholder="JohnCena" label={"First Name"}/>
                <InputBox onChange={(e) => {
                    setLastName(e.target.value)
                }} placeholder="YouCanSeeMe" label={"Last Name"} />
                <InputBox onChange={(e) => {
                    setUsername(e.target.value)
                }} placeholder="nishant1234@gmail.com" label={"Email"}/>
                <InputBox onChange={(e) => {
                    setPassword(e.target.value)
                }} placeholder="1234" label={"password"} />
                <div className="pt-4">
                    <Button onClick={()=>{

axios.post("http://localhost:3000/api/v1/user/signup",{
                            username: username,
                            firstName: firstName,
                            lastName: lastName,
                            password: password
                        })
                    }} label={"Sign up"} />
                </div>
```

```
                <BottomWarning label={"Already have an account?"}
buttonText={"Sign in"} to={"/signin"}/>
            </div>

        </div>

    </div>
}
```

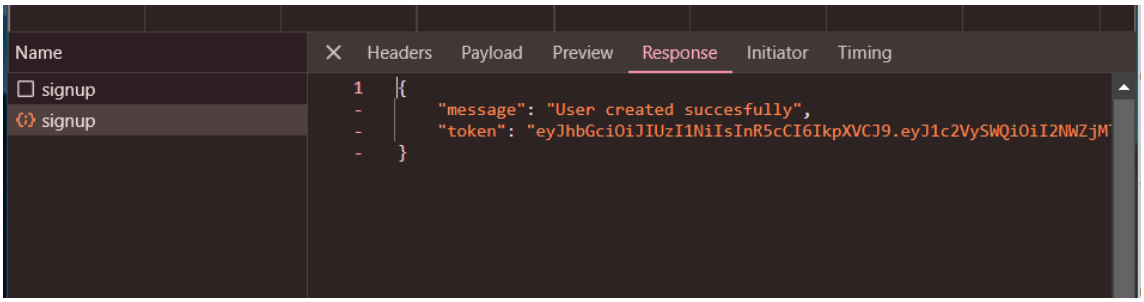
Start both backend and frontend

Lets try to signup the information of user into the database.

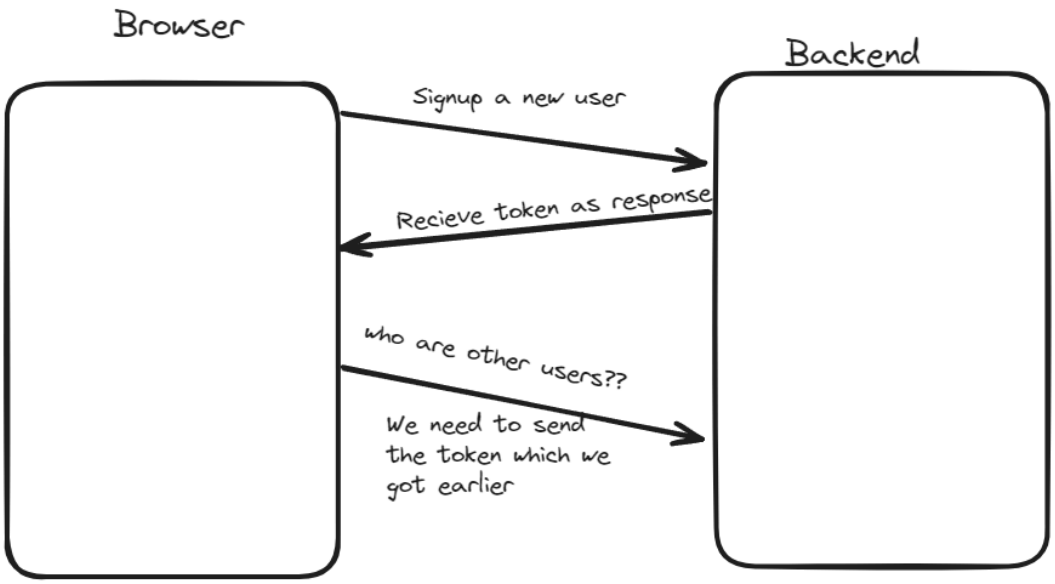




We get back a token



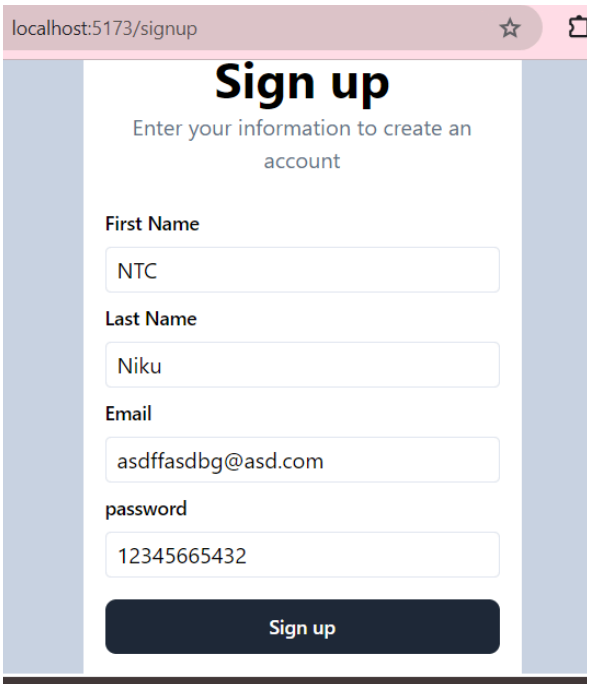
Now the part which we will focus in once we get token as response , where will we store it such that all our future request contains it.

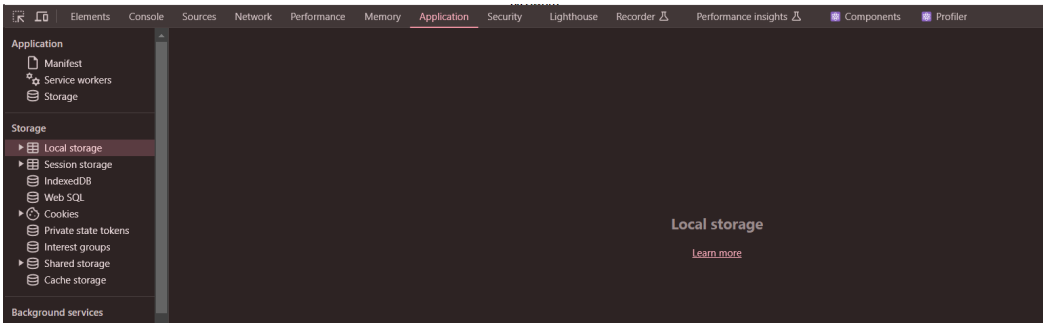


How can we maintain a session, or store the token in Browser
By using localStorage. So that we are persistently login.
(It means our token is stored somewhere)

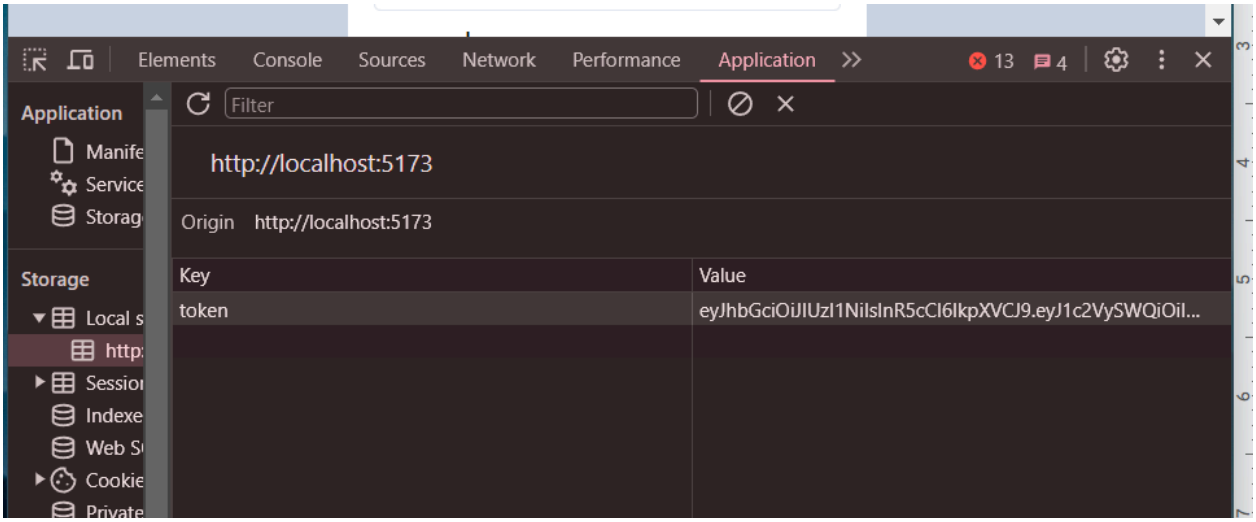
What this will do is that whenever we signup and I get back token, it is stored somewhere in Browser and when We reach /dashboard page then I can just send a request with this token and backend will return all the auth endpoint.
Whenever backend needs an authorization header it will get from the backend.

Now lets signup a new user





We need to make it axios await.
Lets do signup for another user
Now lets check our localStorage



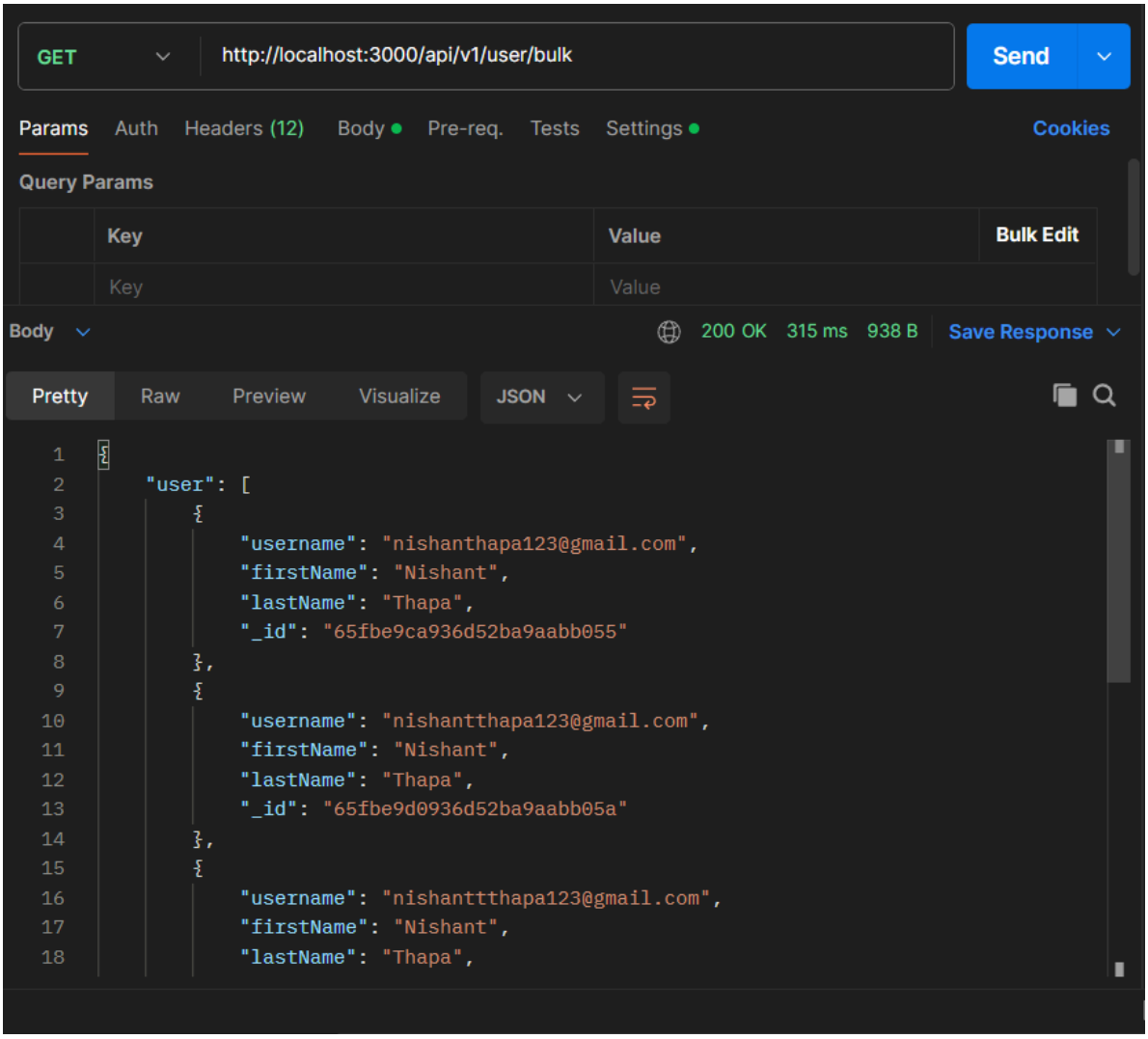
Now we are moving toward authenticated app which can persistently store our session, or keep us logged in unless we log out.

We just remove their authKey(empty their token) whenever they logged out. Now this token is not stored in browser anymore.

The backend will only accept request only when they are sending this token as authorization header in every request.

Let’s see how we can do it.

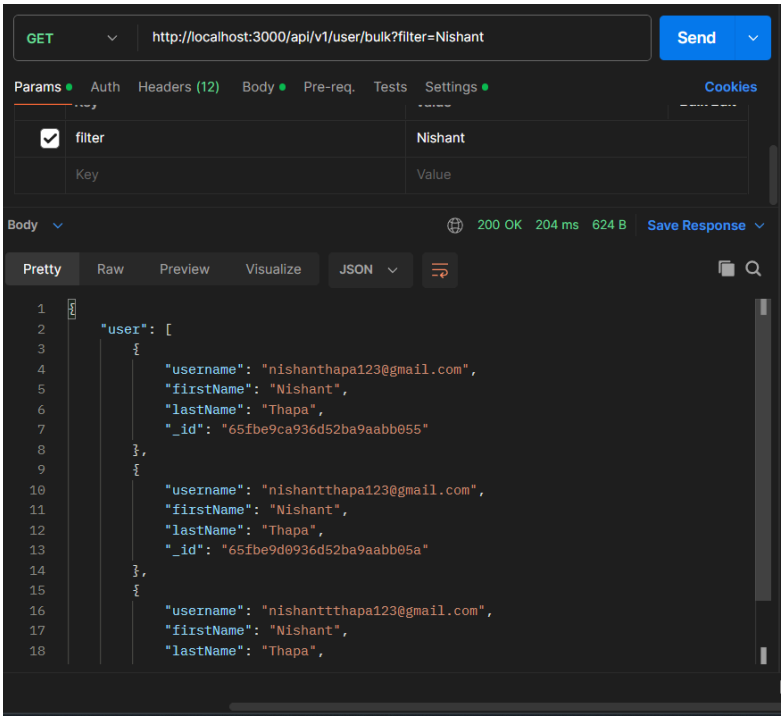
Send the request to get the list of all the user



Only people who are logged in should be able to see the list of users. Currently anyone can see it.

Our backend is already optimized to use filter.

Filtering is on the basis of Firstname or lastname

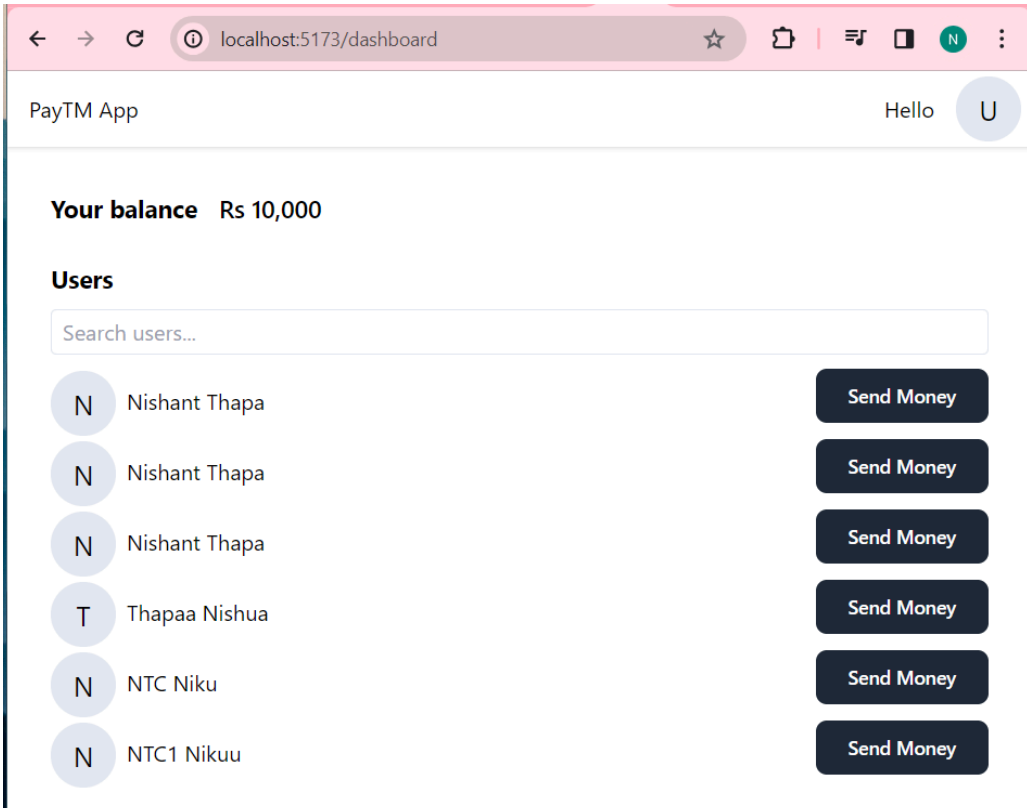


We will work on **User.jsx** currently we have hardcoded the users

```jsx
import { useEffect, useState } from "react"
import { Button } from "./Button"
import axios from "axios";

export const Users = () => {
    // Replace with backend call
    const [users, setUsers] = useState([]);

    useEffect(()=>{
        axios.get("http://localhost:3000/api/v1/user/bulk")
        .then(response => {
            setUsers(response.data.user)
        })
    },[])
    // currently it runs only once

    return <>
        <div className="font-bold mt-6 text-lg">
            Users
        </div>
        <div className="my-2">
```

```
            <input type="text" placeholder="Search users..."
classNameName="w-full px-2 py-1 border rounded border-slate-200"></input>
        </div>
        <div>
            {users.map(user => <User user={user} />)}
        </div>
    </>
}

function User({user}) {
    return <div classNameName="flex justify-between">
        <div classNameName="flex">
            <div classNameName="rounded-full h-12 w-12 bg-slate-200 flex
justify-center mt-1 mr-2">
                <div classNameName="flex flex-col justify-center h-full
text-xl">
                    {user.firstName[0]}
                </div>
            </div>
            <div classNameName="flex flex-col justify-center h-ful">
                <div>
                    {user.firstName} {user.lastName}
                </div>
            </div>
        </div>

        <div classNameName="flex flex-col justify-center h-ful">
            <Button label={"Send Money"} />
        </div>
    </div>
}
```

Currently it is making a axos call only once and we haven’t inserted the filter logic.



We should insert a logic such that backend doesn't return us only.

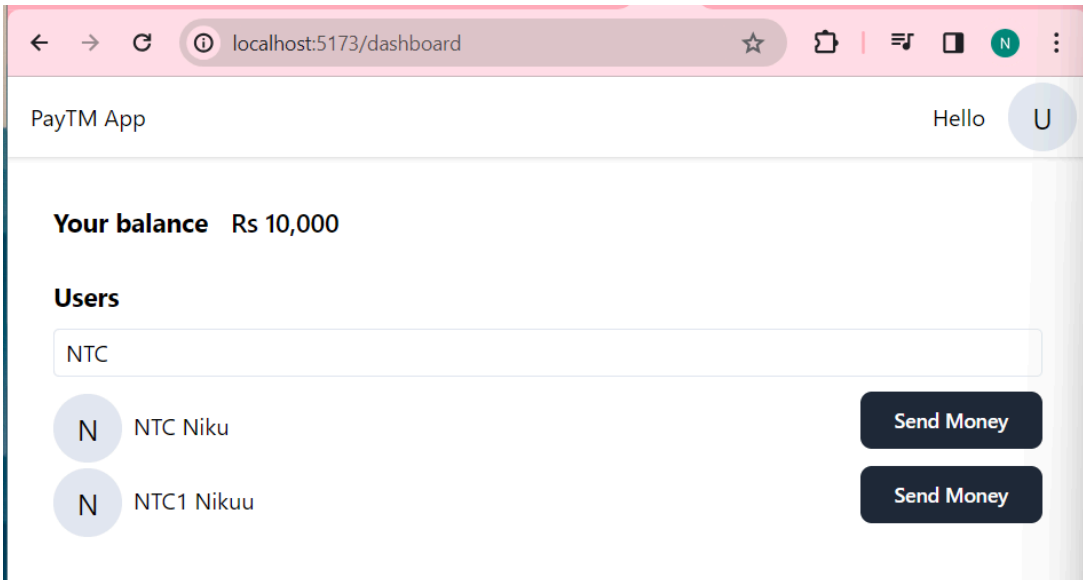
Implementing the filter logic

User.jsx

```
import { useEffect, useState } from "react"
import { Button } from "./Button"
import axios from "axios";

export const Users = () => {
    // Replace with backend call
    const [users, setUsers] = useState([]);
    const [filter,setFilter] = useState("");
    useEffect(()=>{
        axios.get("http://localhost:3000/api/v1/user/bulk?filter="+
filter)
        .then(response => {
            setUsers(response.data.user)
        })
```

```
    },[filter])
    // currently it runs only once
    // in real world we have millions users
    //  we do pagination and backend shows kind of 10 users


    return <>
        <div className="font-bold mt-6 text-lg">
            Users
        </div>
        <div className="my-2">
            <input onChange={(e)=>{
                setFilter(e.target.value)
                // populated the state variable
            }} type="text" placeholder="Search users..." className="w-full
px-2 py-1 border rounded border-slate-200"></input>
        </div>
        <div>
            {users.map(user => <User user={user} />)}
        </div>
    </>
}

function User({user}) {
    return <div className="flex justify-between">
        <div className="flex">
            <div className="rounded-full h-12 w-12 bg-slate-200 flex
justify-center mt-1 mr-2">
                <div className="flex flex-col justify-center h-full
text-xl">
                    {user.firstName[0]}
                </div>
            </div>
            <div className="flex flex-col justify-center h-ful">
                <div>
                    {user.firstName} {user.lastName}
                </div>
            </div>
        </div>
```

```
        <div className="flex flex-col justify-center h-ful">
            <Button label={"Send Money"} />
        </div>
    </div>
}
```

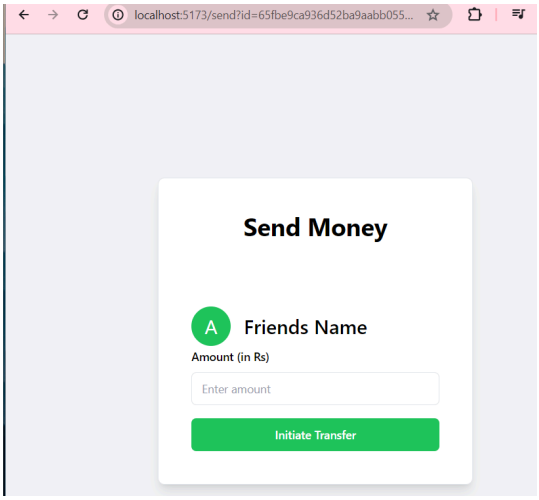


We should add debouncing

Now we need to implement the Send Money button as we click on the button it should redirect to the another page where we will implement its functionality.

Button is now working.

```
<div className="flex flex-col justify-center h-ful">
            <Button onClick={(e)=>{
                navigate("/send?id="+user._id+"&name"+user.firstName)
                // We can also use LINK
            }} label={"Send Money"} />
        </div>
```



Now i need to render name and amount.

The thing we need to know is how to get access to query parameter

```
import { useSearchParams } from "react-router-dom"
import axios from "axios";
import { useState } from "react";

export const SendMoney = () => {
    const [searchParams] = useSearchParams();
    const id = searchParams.get("id")
    const name = searchParams.get("name");
    const [amount,setAmount] = useState(0);

    return <div className="flex justify-center h-screen bg-gray-100">
        <div className="h-full flex flex-col justify-center">
            <div
                className="border h-min text-card-foreground max-w-md p-4
space-y-8 w-96 bg-white shadow-lg rounded-lg"
            >
                <div className="flex flex-col space-y-1.5 p-6">
                <h2 className="text-3xl font-bold text-center">Send
Money</h2>
                </div>
                <div className="p-6">
                <div className="flex items-center space-x-4">
                    <div className="w-12 h-12 rounded-full bg-green-500
flex items-center justify-center">
```

```
                    <span className="text-2xl
text-white">{name[0].toUpperCase()}</span>
                    </div>
                    <h3 className="text-2xl font-semibold">{name}</h3>
                </div>
                <div className="space-y-4">
                    <div className="space-y-2">
                    <label
                        className="text-sm font-medium leading-none
peer-disabled:cursor-not-allowed peer-disabled:opacity-70"
                        htmlFor="amount"
                    >
                        Amount (in Rs)
                    </label>
                    <input
                        onChange={(e) => {
                            setAmount(e.target.value)
                        }}
                        type="number"
                        className="flex h-10 w-full rounded-md border
border-input bg-background px-3 py-2 text-sm"
                        id="amount"
                        placeholder="Enter amount"
                    />
                    </div>
                    <button onClick={()=>{

axios.post("http://localhost:3000/api/v1/account/transfer",{
                            to:id,
                            amount
                        },{
                            headers:{
                                Authorization:"Bearer " +
localStorage.getItem("token")
                                // if token not , then it is logged out
                            }
                        })
                    }} className="justify-center rounded-md text-sm
font-medium ring-offset-background transition-colors h-10 px-4 py-2 w-full
bg-green-500 text-white">
```

```
                    Initiate Transfer
                </button>
            </div>
            </div>
        </div>
    </div>
</div>
}
```