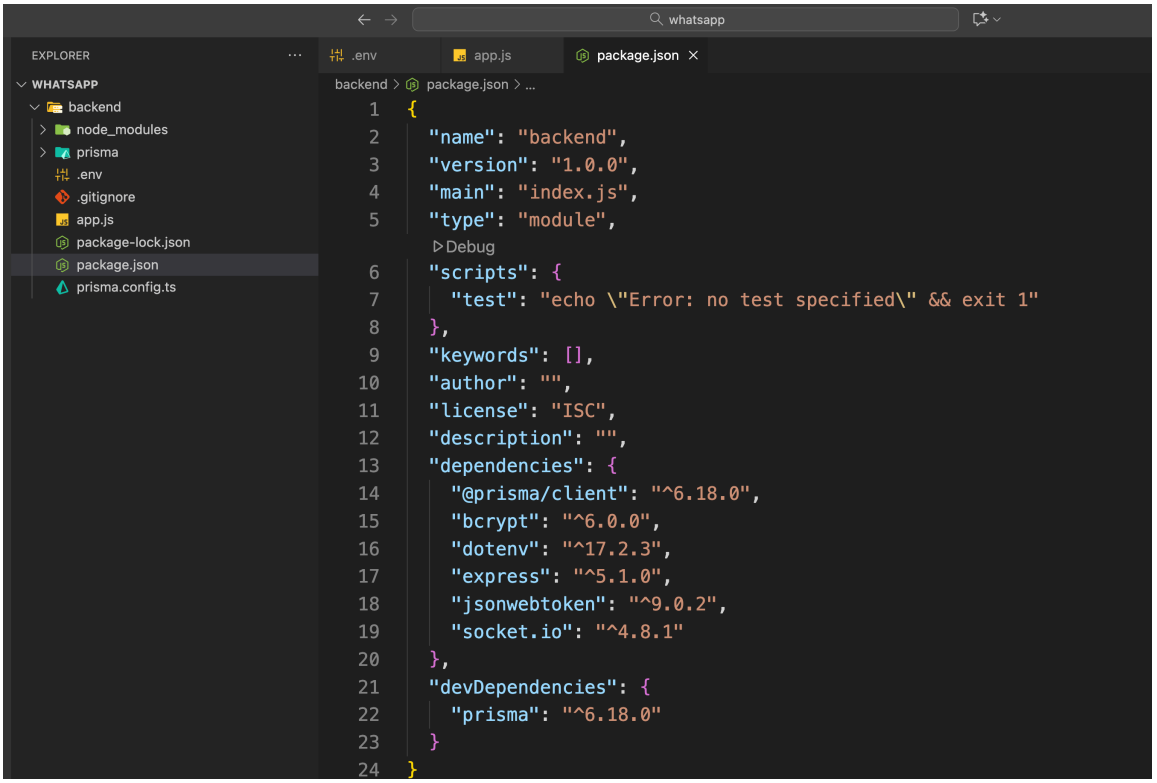
App SETUP

BACKEND - CODE

▼ Terminal Commands

```
// Make a folder
mkdir backend
cd backend

// initialise the package.json
npm init -y

// install express and prisma
npm i express dotenv jsonwebtoken bcrypt socket.io @prisma/client co
rs
npm install prisma -D // install it as a dev dependency, not to be used in
production code

// Initialise the prisma
npx prisma init --datasource-provider postgresql --output ../generated/
prisma
```

Also set `type: module` inside the package.json to use `import` and `export` in nodejs project.



▼ Start Building Express and Socket-io App Together

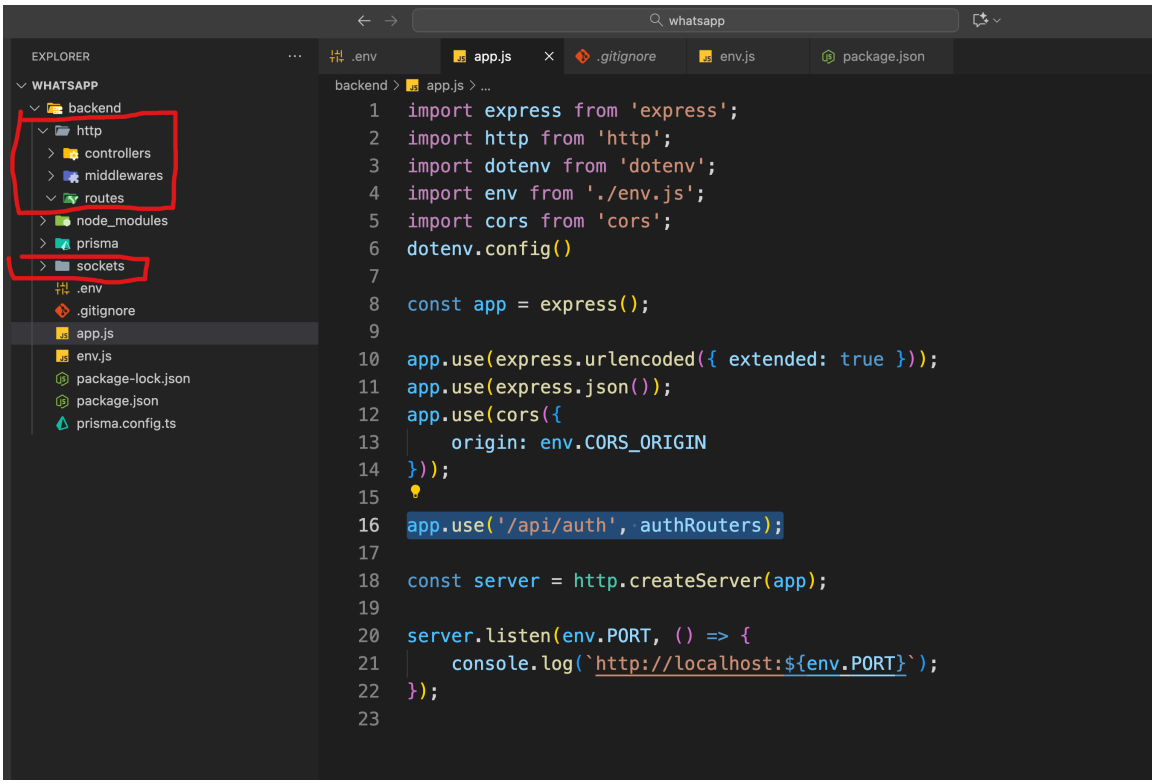
```
// Setup your express and socket app together, we can keep them in se
parate folders too
import express from 'express';
import http from 'http';
import dotenv from 'dotenv';
import env from './env.js';
import cors from 'cors';
dotenv.config()

const app = express();

app.use(express.urlencoded({ extended: true }));
app.use(express.json());
app.use(cors({
   origin: env.CORS_ORIGIN
}));


const server = http.createServer(app);
```

```
server.listen(env.PORT, () => {
    console.log(`http://localhost:${env.PORT}`);
});
```

ALL SET TO USE EXPRESS AND SOCKETS BOTH NOW.....

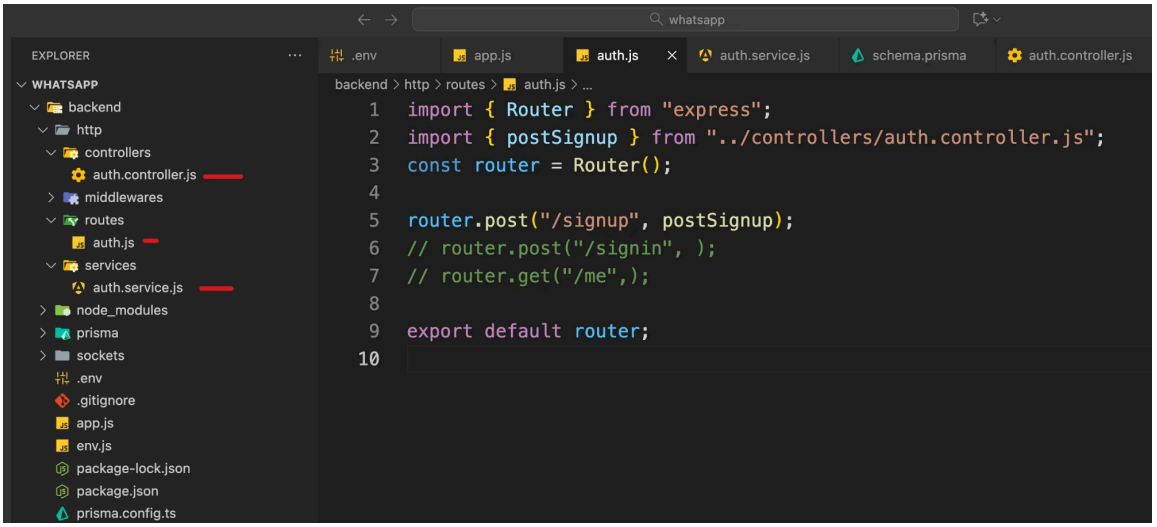
▼ Setup - Signup code on `/api/auth`

Now, we would need to setup `http` and `socket` currently they are empty for now..



Let's build `signup` code now.

We are going to write the code here in the files mentioned below:



Inside: `/routes/auth.js`

```
import { Router } from "express";
import { postSignup } from "../controllers/auth.controller.js";
const router = Router();

router.post("/signup", postSignup);
// router.post("/signin", );
// router.get("/me",);

export default router;
```

Now, we need to setup `model` to store signup information of the user

Inside: `schema.prisma`

```
model User {
  id        String   @id @default(uuid()) @db.Uuid
  email     String   @unique
  name      String?
  password  String
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}
```

ADD THE FOLLOWING COMMAND at the top of `prisma.config.ts`

```
import "dotenv/config"; // This helps to use the .env file variables in this file
```

Now, do migration to create the table inside the postgres

```
npx prisma generate // This will create a generated/prisma that we can
// use to run the prisma commands
npx prisma migrate dev --name init // To load the model into the databa
se
```

Inside: `controllers/auth.controller.js`

```
import { signup } from "../services/auth.service.js";

export async function postSignup(req, res, next) {
   try {
      const { email, name, password } = req.body;

      let data = await signup({ email, name, password });
      res.status(200).json(data)

   } catch (error) {
      res.status(500).json({
         error,
         message: 'Signup Failed'
      })
   }
}
```

Inside: `services/auth.service.js`

```
import env from "../../env.js";
import jwt from 'jsonwebtoken';
import { PrismaClient } from "@prisma/client";
import bcrypt from 'bcrypt';
let prisma = new PrismaClient();

function signToken(payload) {
```

```
    return jwt.sign(payload, env.JWT_SECRET);
}

async function signup({ email, password, name }) {
    const existing = await prisma.user.findUnique({ where: { email } });

    if (existing) {
        const err = new Error('Email already in use');
        err.status = 400;
        throw err;
    }

    try {
        const salt = await bcrypt.genSalt(10);
        const passwordHash = await bcrypt.hash(password, salt);

        const user = await prisma.user.create({ data: { email, name, passw
ord: passwordHash } });
        const token = signToken({ id: user.id, email: user.email });
        console.log(token)
        return { user: { id: user.id, email: user.email, name: user.name }, tok
en };
    } catch (error) {
        console.log(error)
        throw error;
    }
}

export { signup }
```

▼ Setup - Signin code on `/api/auth`

It is really simple now,

Inside `routes/auth.js`

```
import { Router } from "express";
import { postSignin, postSignup } from "../controllers/auth.controller.js";
const router = Router();
```

```
router.post("/signup", postSignup);
router.post("/signin", postSignin);
// router.get("/me",);

export default router;
```

Inside `controllers/auth.controller.js`

```
import { signin, signup } from "../services/auth.service.js";

export async function postSignup(req, res, next) {
    // SAME
}

export async function postSignin(req, res, next) {
  try {
    const { email, password } = req.body;
    const result = await signin({ email, password });

    res.status(200).json(result);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
}
```

Inside `services/auth.service.js`

```
import env from "../../env.js";
import jwt from 'jsonwebtoken';
import { PrismaClient } from "@prisma/client";
import bcrypt from 'bcrypt';
let prisma = new PrismaClient();

function signToken(payload) {
  // SAME
}

async function signup({ email, password, name }) {
```

```
    // SAME
}

async function signin({ email, password }) {

  const user = await prisma.user.findUnique({ where: { email } });

  if (!user) {
    const err = new Error('Invalid credentials');
    err.status = 401;
    throw err;
  }

  const ok = await bcrypt.compare(password, user.password);

  if (!ok) {
    const err = new Error('Invalid credentials');
    err.status = 401;
    throw err;
  }

  const token = signToken({ id: user.id, email: user.email });

  return { user: { id: user.id, email: user.email, name: user.name }, toke
n };
}

export { signup, signin }
```

▼ ADDING ZOD for checking input

```
import { z } from "zod";

export const signupSchema = z.object({
  email: z.string().trim().toLowerCase().email("Invalid email format"),
  password: z
    .string()
    .min(8, "Password must be at least 8 characters long")
```

```
      .max(64, "Password too long"),
  name: z
      .string()
      .trim()
      .min(1, "Name cannot be empty")
      .max(100, "Name too long")
      .optional(),
});

export const signinSchema = z.object({
  email: z.string().trim().toLowerCase().email("Invalid email format"),
  password: z
      .string()
      .min(8, "Password must be at least 8 characters long")
      .max(64, "Password too long"),
});
```

Updating the `controllers/auth.controller.js`

```
import { signupSchema } from "../schemas/auth.schema.js";
import { signin, signup } from "../services/auth.service.js";

export async function postSignup(req, res, next) {
  try {

      const result = signupSchema.safeParse(req.body);
      if (!result.success) {
          return res.status(400).json({
              error: 'Signup Validation failed'
          });
      }
      const { email, name, password } = result.data;
      let data = await signup({ email, name, password });

      res.status(200).json(data)
  } catch (error) {
      console.log(error)
      res.status(500).json({
```

```
            error,
            message: 'Signup Failed'
        })
    }
}

export async function postSignin(req, res, next) {
    try {

        const result = signupSchema.safeParse(req.body);

        if (!result.success) {

            return res.status(400).json({
                error: 'Signin Validation failed',
                details: JSON.parse(result.error.message)
            });
        }

        const { email, password } = result.data;
        const data = await signin({ email, password });

        res.status(200).json(data);
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
}
```

▼ If already loggedIn then fetching the user details using JWT

Adding to `routes/auth.js`

```
import { Router } from "express";
import { getMe, postSignin, postSignup } from "../controllers/auth.contr
oller.js";
import { requireAuth } from "../middlewares/requireAuth.js";

const router = Router();
```

```
router.post("/signup", postSignup);
router.post("/signin", postSignin);
router.get("/me", requireAuth, getMe);

export default router;
```

Inside `requireAuth.js`

```
import jwt from "jsonwebtoken";
import { PrismaClient } from '@prisma/client';
let prisma = new PrismaClient();

const { JWT_SECRET } = process.env;

export const requireAuth = async (req, res, next) => {
  try {
    const header = req.headers.authorization || "";
    const token = header.startsWith("Bearer ") ? header.split(" ")[1] : n
ull;
    if (!token) return res.status(401).json({ error: "Token required" });

    const decoded = jwt.verify(token, JWT_SECRET);
    const user = await prisma.user.findUnique({
      where: { id: decoded.id }
    });
    if (!user) return res.status(401).json({ error: "Invalid or expired toke
n" });

    req.user = user;
    next();
  } catch (err) {
    console.error(err);
    return res.status(401).json({ error: "Unauthorized" });
  }
};
```

FRONTEND - CODE for SIGNUP

▼ Build the react app

```
mkdir client
cd client
npm create vite@latest // Assuming you know that you know to create a
pp with JS
```

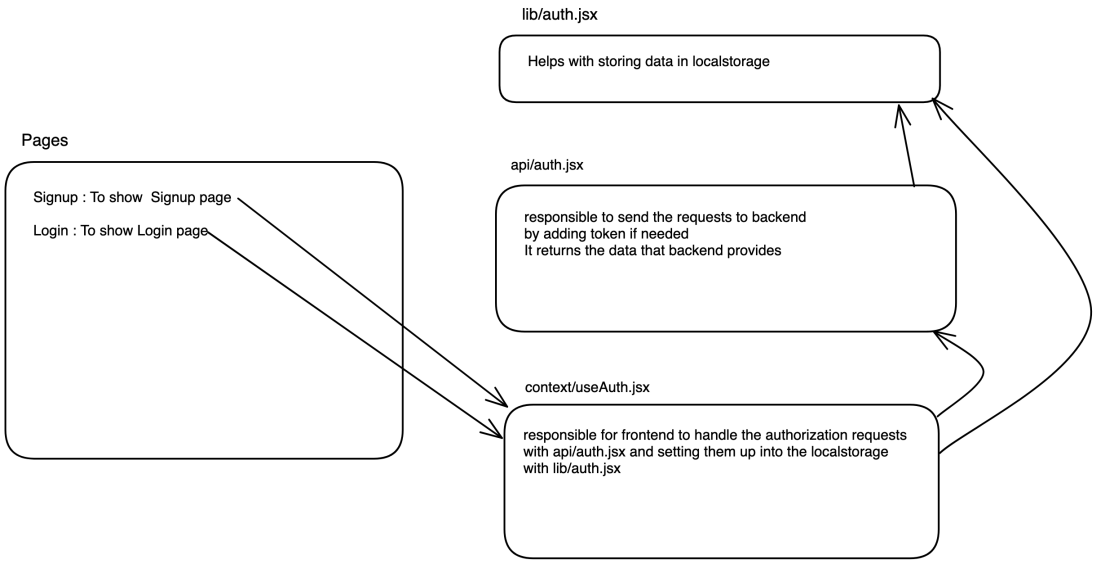
Installing the react-router and setting it up

```
npm i react-router
```

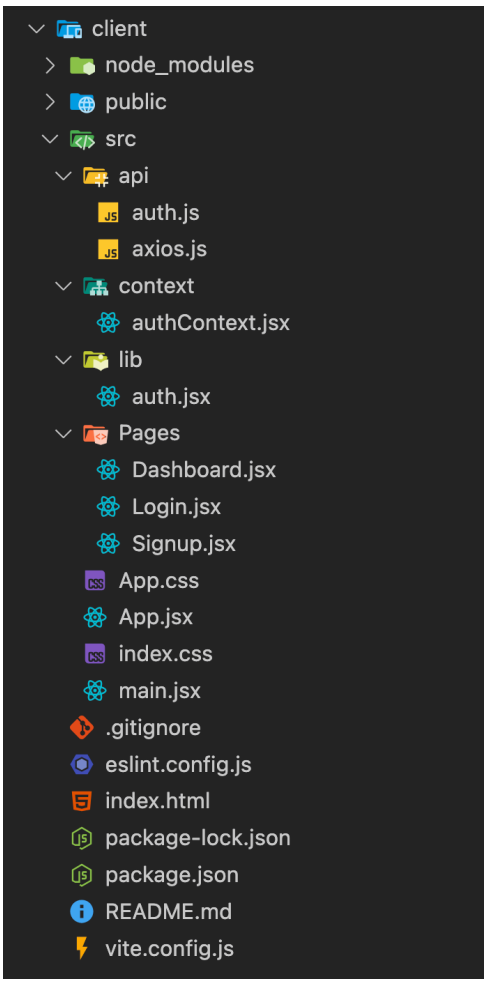
Updating `App.jsx`

```
import { createRoot } from "react-dom/client";
import "./index.css";
import App from "./App.jsx";
import { BrowserRouter } from "react-router";

createRoot(document.getElementById("root")).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

What we are looking to create is 👇🏻



Folder Structure looks like this:



Now, lets create a way to

- store data inside `localstorage`
- sending request and automatically sending token inside the headers with every request

First thing would be handled by `lib/auth.js`

```
export default {
  get token() { return localStorage.getItem('token'); },
  set token(v) { v ? localStorage.setItem('token', v) : localStorage.remo
veItem('token'); },
  get user() {
    const user = localStorage.getItem('user');
    return user ? JSON.parse(user) : null;
  },
```

```
  set user(u) {
    u ? localStorage.setItem('user', JSON.stringify(u)) : localStorage.re
moveItem('user');
  },
  logout() { this.token = null; this.user = null; }
};
// To set/get user and token inside the localstorage
```

`Signup.jsx` inside the pages

```
import { useEffect, useState } from "react";
import { Link, useNavigate } from "react-router";
import auth from "../lib/auth";
import { authApi } from "../api/auth";

export default function Signup() {
 const [name, setName] = useState("");
 const [email, setEmail] = useState("");
 const [pwd, setPwd] = useState("");
 const [err, setErr] = useState("");
 const navigate = useNavigate();

 useEffect(() => {
  if (auth.token) {
   authApi
    .me()
    .then(() => {
     navigate("/dashboard");
    })
    .catch(() => auth.logout());
  }
 }, [navigate]);

 async function handleSubmit(e) {
  e.preventDefault();
  setErr("");
  try {
   const { user, token } = await authApi.signup({
```

```
      email,
      password: pwd,
      name,
    });
    auth.token = token;
    auth.user = user;
    setEmail("");
    setPwd("");
    setName("");
    navigate("/dashboard");
  } catch (e) {
    setErr(e.message);
  }
}

return (
  <div>
    <h2>Create account</h2>
    <form onSubmit={handleSubmit}>
      <input
        placeholder="Name (optional)"
        value={name}
        onChange={(e) => setName(e.target.value)}
      />
      <input
        placeholder="Email"
        type="email"
        autoComplete="username"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        required
      />
      <input
        placeholder="Password"
        type="password"
        autoComplete="current-password"
        value={pwd}
        onChange={(e) => setPwd(e.target.value)}
```

```
          required
        />
        <button type="submit">Sign up</button>
      </form>
      {err && <p>{err}</p>}
      <p>
        Already have an account? <Link to="/signin">Sign in</Link>
      </p>
    </div>
  );
}
```

Inside `Login.jsx`

```
import { useState } from "react";
import { Link, useNavigate } from "react-router";
import { useAuth } from "../context/authContext";

export default function Signin() {
  const [email, setEmail] = useState("");
  const [pwd, setPwd] = useState("");
  const [err, setErr] = useState("");
  const { signin } = useAuth();
  const navigate = useNavigate();

  async function handleSubmit(e) {
    e.preventDefault();
    setErr("");
    try {
      await signin({ email, password: pwd });
      navigate("/dashboard");
    } catch (e) {
      setErr(e.message);
    }
  }

  return (
    <div>
```

```
      <h2>Sign in</h2>
      <form onSubmit={handleSubmit}>
        <input
          placeholder="Email"
          type="email"
          autoComplete="username"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          required
        />
        <input
          placeholder="Password"
          type="password"
          autoComplete="current-password"
          value={pwd}
          onChange={(e) => setPwd(e.target.value)}
          required
        />
        <button type="submit">Sign in</button>
      </form>
      {err && <p>{err}</p>}
      <p>
        No account? <Link to="/signup">Create one</Link>
      </p>
    </div>
  );
}
```

Inside `context/authContext.jsx`

```
import { createContext, useContext, useEffect, useState } from "react";
import { authApi } from "../api/auth";
import auth from "../lib/auth";

const AuthContext = createContext();

export function AuthProvider({ children }) {
  const [user, setUser] = useState(auth.user);
```

```
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    if (auth.token) {
      authApi
        .me()
        .then(({ user }) => {
          auth.user = user;
          setUser(user);
        })
        .catch(() => {
          auth.logout();
          setUser(null);
        })
        .finally(() => setLoading(false));
    } else {
      setLoading(false);
    }
  }, []);

  const signin = async ({ email, password }) => {
    const { user, token } = await authApi.signin({ email, password });
    // Setting the token inside the localstorage too along with user
    auth.token = token;
    auth.user = user;
    setUser(user);
    return { user, token };
  };

  const signup = async ({ email, password, name }) => {
    const { user, token } = await authApi.signup({ email, password, name
});
    // Setting the token inside the localstorage too along with user
    auth.token = token;
    auth.user = user;
    setUser(user);
    return { user, token };
  };
```

```
  const logout = () => {
    auth.logout();
    setUser(null);
  };

  return (
    <AuthContext.Provider
      value={{
        user,
        isLoggedIn: !!user,
        loading,
        signin,
        signup,
        logout,
        token: auth.token || null,
      }}
    >
      {children}
    </AuthContext.Provider>
  );
}

export function useAuth() {
  return useContext(AuthContext);
}
```

Inside `api/auth.jsx`

```
import axios from "./axios";

export async function api(path, { method = 'GET', body, auth = false })
{
    const headers = { 'Content-Type': 'application/json' };
    if (auth) {
        const t = localStorage.getItem('token');
        if (t) headers.Authorization = `Bearer ${t}`;
    }
```

```
    const { data } = await axios({
        method,
        url: `${path}`,
        headers,
        data: body ? body : undefined,
    });

    return data;
}

export const authApi = {
    async signup({ email, password, name }) {
        return api('/api/auth/signup', { method: 'POST', body: { email, pass
word, name } });
    },
    async signin({ email, password }) {
        return api('/api/auth/signin', { method: 'POST', body: { email, pass
word } });
    },
    async me() {
        return api('/api/auth/me', { auth: true });
    }
};
```

Inside `api/axios.jsx` also do `npm i axios` before we can use this axios package....

```
import axios from 'axios';

export default axios.create({
    baseURL: 'http://localhost:4444'
});

// run npm i axios before we are able to use this package...
```

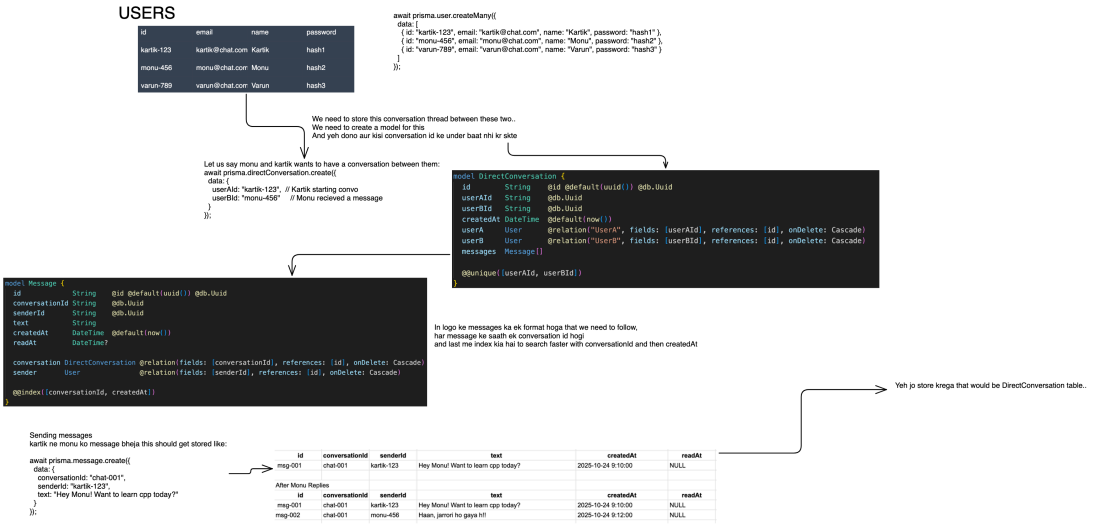
Inside `Dashboard.jsx`:

```
import React from 'react'
import auth from '../lib/auth'

const Dashboard = () => {
  return (
    <div>
      Welcome to the Dashboard: {auth.user}
    </div>
  )
}

export default Dashboard
```

BACKEND - Chat Database application

▼ Building the Database for Chat Application between two people...

It will look something like this



Final Users Model

```
model User {
  id        String   @id @default(uuid()) @db.Uuid
```

```
  email     String   @unique
  name      String?
  password  String
  avatar    String?
  status    String?
  isOnline  Boolean  @default(false)
  lastSeen  DateTime @default(now())
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  conversationsA DirectConversation[] @relation("UserA")
  conversationsB DirectConversation[] @relation("UserB")
  messages       Message[]

  @@index([email])
  @@index([isOnline])
}
```

▼ What is @db.Uuid

- Maps to PostgreSQL's native `UUID` type instead of `TEXT` as column
- Type safety

Direct Conversation

```
model DirectConversation {
  id        String   @id @default(uuid()) @db.Uuid
  userAId   String   @db.Uuid
  userBId   String   @db.Uuid
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  userA    User      @relation("UserA", fields: [userAId], references: [id], o
nDelete: Cascade)
  userB    User      @relation("UserB", fields: [userBId], references: [id],
onDelete: Cascade)
  messages Message[]

  @@unique([userAId, userBId])
```

```
  @@index([userAId])
  @@index([userBId])
  @@index([updatedAt])
}
```

Messages

```
model Message {
  id             String      @id @default(uuid()) @db.Uuid
  conversationId String      @db.Uuid
  senderId       String      @db.Uuid
  text           String      @db.Text
  type           MessageType @default(TEXT)
  isRead         Boolean     @default(false)
  isDelivered    Boolean     @default(false)
  createdAt      DateTime    @default(now())
  updatedAt      DateTime    @updatedAt

  editedAt  DateTime?
  isDeleted Boolean   @default(false)

  conversation DirectConversation @relation(fields: [conversationId], ref
erences: [id], onDelete: Cascade)
  sender       User               @relation(fields: [senderId], references: [id],
onDelete: Cascade)

  @@index([conversationId, createdAt])
  @@index([senderId])
}

enum MessageType {
  TEXT
  IMAGE
  FILE
}
```

RUN command

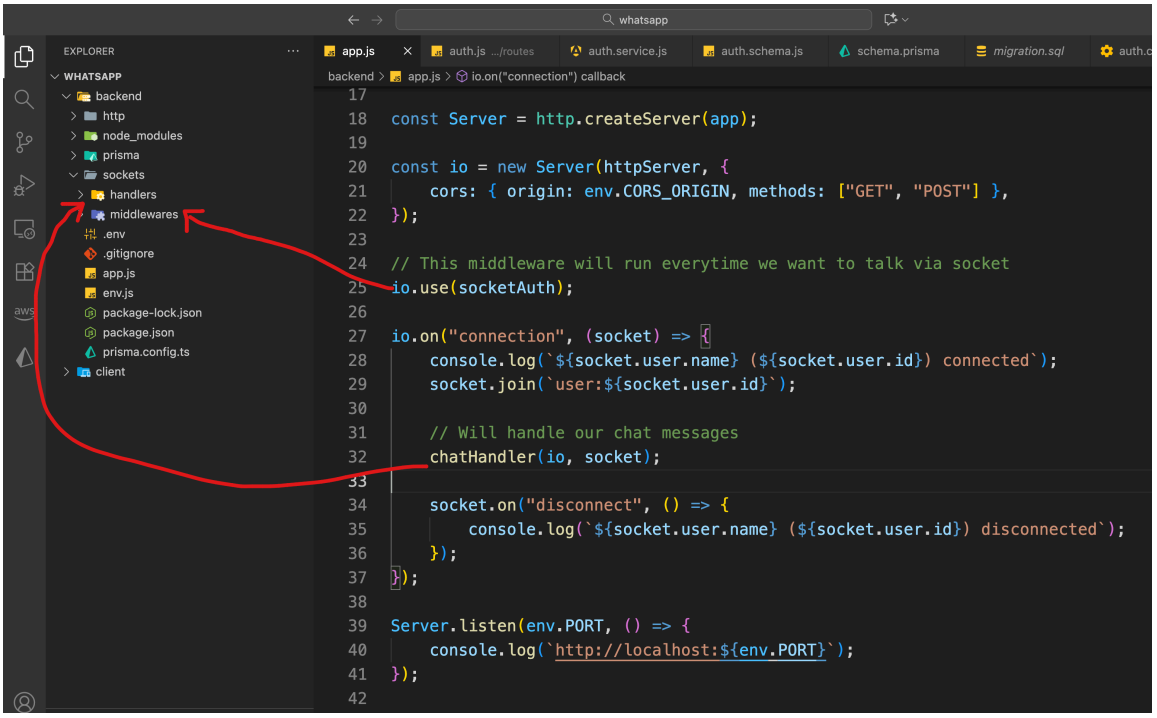
```
npx prisma migrate dev --name message_directcoversation_models
```

▼ Adding SOCKET.io to our backend

Update app.js

```
import express from 'express';
import http from 'http';
import dotenv from 'dotenv';
dotenv.config()
import env from './env.js';
import cors from 'cors';
import authRouters from './http/routes/auth.js';
const app = express();

app.use(express.urlencoded({ extended: true }));
app.use(express.json());
app.use(cors({
   origin: env.CORS_ORIGIN
}));

app.use('/api/auth', authRouters);

const Server = http.createServer(app);

const io = new Server(httpServer, {
   cors: { origin: env.CORS_ORIGIN, methods: ["GET", "POST"] },
});

// This middleware will run everytime we want to talk via socket
io.use(socketAuth);

io.on("connection", (socket) => {
   console.log(`${socket.user.name} (${socket.user.id}) connected`);
   socket.join(`user:${socket.user.id}`);

   // Will handle our chat messages
```

```
    chatHandler(io, socket);

    socket.on("disconnect", () => {
        console.log(`${socket.user.name} (${socket.user.id}) disconnecte
d`);
    });
});

Server.listen(env.PORT, () => {
    console.log(`http://localhost:${env.PORT}`);
});
```



Inside `middlewares` create `socket.auth.js`

```
import jwt from "jsonwebtoken";
import { PrismaClient } from "@prisma/client";
import env from "../../env";
let prisma = new PrismaClient()

const { JWT_SECRET } = env;

export async function socketAuth(socket, next) {
```

```
    try {
      // This will be sent from the client side
      /*
      // This is how we can send it from the users end...
       const s = io("http://localhost:4444", {
         auth: { token },
      });
      */
      const token =
        socket.handshake.auth?.token ||
        socket.handshake.headers?.authorization?.split(" ")[1];

      if (!token) throw new Error("Token missing");

      const payload = jwt.verify(token, JWT_SECRET);
      const userId = payload.id;

      if (!userId) throw new Error("Invalid token payload");

      const user = await prisma.user.findUnique({
        where: { id: userId },
        select: { id: true, email: true, name: true },
      });

      if (!user) throw new Error("User not found");

      // Now to use it in socket entire app we can do this
      socket.user = user;
      next();
    } catch (err) {
      console.error("Socket auth failed:", err.message);
      next(new Error("Unauthorized"));
    }
}
```

Create `handlers/chathandler.js`

```
import { PrismaClient } from "@prisma/client";
import { getOrCreateConversation } from "../utils/conversation.js";

let prisma = new PrismaClient();

export async function getOrCreateConversation(userAId, userBId) {
  const [a, b] = [userAId, userBId].sort();

  let conv = await prisma.directConversation.findUnique({
    where: { userAId_userBId: { userAId: a, userBId: b } },
  });

  if (!conv) {
    conv = await prisma.directConversation.create({
      data: { userAId: a, userBId: b },
    });
  }

  return conv;
}


export function chatHandler(io, socket) {
    // If a user sends a message then this event will get triggered...
  socket.on("chat:send", async (payload, cb) => {
    try {
      const { receiverId, text } = payload;
      if (!receiverId || !text?.trim()) throw new Error("Invalid data");

      const conv = await getOrCreateConversation(socket.user.id, rec
eiverId);

      const message = await prisma.message.create({
        data: {
          conversationId: conv.id,
          senderId: socket.user.id,
          text: text.trim(),
        },
```

```
            include: {
                sender: { select: { id: true, name: true, email: true } },
            },
        });

        io.to(`user:${receiverId}`).emit("chat:new", message);
        io.to(`user:${socket.user.id}`).emit("chat:new", message);

        cb?.({ ok: true, message });
    } catch (err) {
        console.error("chat:send error:", err.message);
        cb?.({ ok: false, error: err.message });
    }
  });

  socket.on("chat:history", async ({ withUserId, limit = 50 }, cb) => {
    try {
        const conv = await getOrCreateConversation(socket.user.id, wit
hUserId);
        const messages = await prisma.message.findMany({
            where: { conversationId: conv.id },
            orderBy: { createdAt: "asc" },
            take: Number(limit),
            include: { sender: true },
        });
        // This call backs will be sending the messages to the frontend
        cb?.({ ok: true, messages });
    } catch (err) {
        cb?.({ ok: false, error: err.message });
    }
  });
}
```

▼ Building the frontend inside `Dashboard.jsx`

```
import { useEffect, useState } from "react";
import { useAuth } from "../context/AuthContext";
import { io } from "socket.io-client";
```

```
import auth from "../lib/auth";

export default function Dashboard() {
  const { user, logout } = useAuth();
  const token = auth.token;
  const [socket, setSocket] = useState(null);
  const [isConnected, setIsConnected] = useState(false);
  const [receiverId, setReceiverId] = useState("");
  const [text, setText] = useState("");
  const [messages, setMessages] = useState([]);

  useEffect(() => {
    if (!token || socket) return;
    const s = io("http://localhost:4444", {
      auth: { token },
    });
    setSocket(s);

    s.on("connect", () => {
      console.log("Connected");
      setIsConnected(true);
    });

    s.on("disconnect", () => {
      console.log("Disconnected");
      setIsConnected(false);
    });

    s.on("chat:new", (msg) => setMessages((prev) => [msg, ...prev]));
    s.on("chat:new-batch", (msgs) => setMessages((prev) => [...prev, ...m
sgs]));

    return () => s.disconnect();
  }, [token]);

  const sendMessage = () => {
    if (!isConnected) {
      alert("Socket not connected yet. Please wait a moment.");
```

```
    return;
  }
  if (!text.trim() || !receiverId.trim()) return;

  socket.emit("chat:send", { receiverId, text }, (res) => {
    if (!res.ok) alert(res.error);
    setText("");
  });
};

return (
  <div>
    <h2>Welcome, {user?.name || user?.email}</h2>
    <p>Status: {isConnected ? "Connected" : "Connecting..."}</p>
    <button onClick={logout}>Logout</button>

    <div>
      <h4>Send Message (even if user is offline)</h4>
      <input
        placeholder="Enter Receiver ID"
        value={receiverId}
        onChange={(e) => setReceiverId(e.target.value)}
      />
      <br />
      <input
        placeholder="Message"
        value={text}
        onChange={(e) => setText(e.target.value)}
      />
      <br />
      <button onClick={sendMessage} disabled={!isConnected}>
        Send
      </button>
    </div>

    <div>
      <h4>All Messages</h4>
      <div>
```

```
        {messages.map((m, i) ⇒ (
          <div key={i}>
            <span>
              {m.text}
            </span>
          </div>
        ))}
      </div>
    </div>
  </div>
 );
}
```