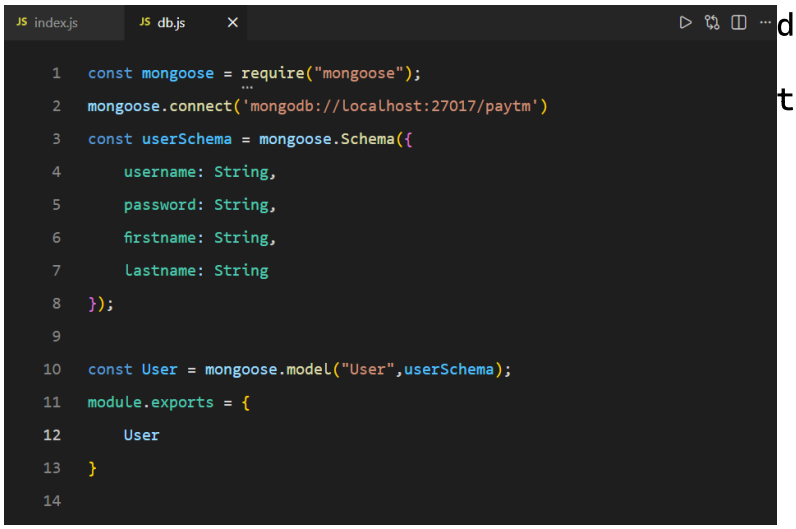
First using the docker file
first off check in the docker folder or not
docker build ./ -t mongodb:4.7-replset
then to run it use
docker run mongodb:4.7-replset
docker run -d -p 27017:27017 mongodb:4.7-replset

use this one to run at a specific port
docker run --name mongodb-replset22 -p 27017:27017 -d mongodb:4.7-replset

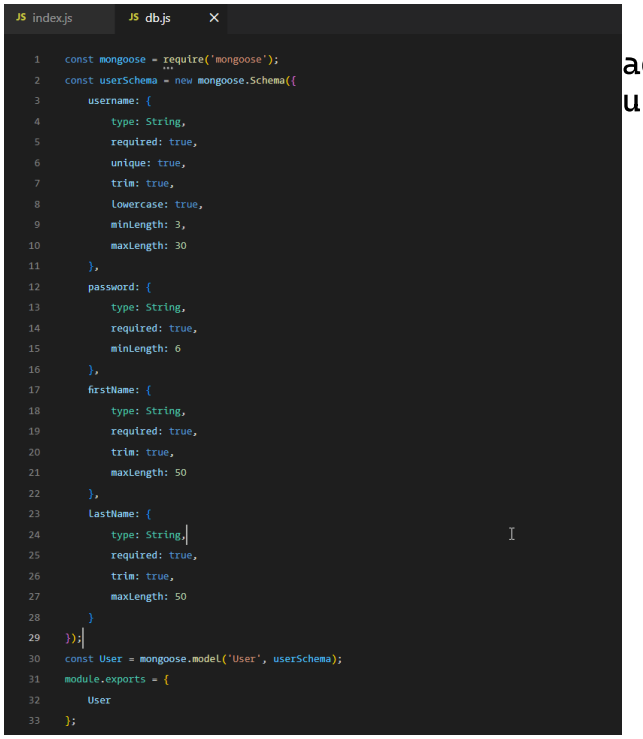
docker ps - to view the instances running
docker kill container_id - to kill the instances

then just try connecting with the default id in the mongod db



define a mongoose schema

this is easy way of doing



adding constraints in the schema like the username should be unique , lowercase etc ...

so what we are doing is setting up the routes thing we are just defining the v1 in the main index.js and then routing the other routes in the index.js of the routes folder and trying to run that

its like firstly the request goes to backend index.js which goes to mainrouter in the router folder like
//api/v1/user
and then from the index.js of the routes folder it goes to
//api/v1/user/signin

installing libraries
npm I cors

in user.js what all things we are doing are

first describing the zod schema
for signup endpoint

```
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
    })
```

checking for the data user send was correct or not

here we are like checking if the email is present in the db or not and then confiming that

```
    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.json({
        message: "User created successfully",
        token: token
    })
})
```

if not found then inserting the data

assigning a jwt and then assigning to the username

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


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

module.exports = router;
```

first tried to check again

if the user is found checking with the jwt token if its matched the user will be signed in

ZOD
User sends data → Zod **verifies it** → then you use it

zod helps to put many checks and all in the data received

authenticator does – stateless otps

```
import { user } from "./db";
const express = require("express");
const cors = require("cors");

app.use(cors());
app.use(express.json());

const mainRouter = require("./routes/index");

const app = express();

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

the CORS must be above the routes you are using because the express and cors will not run then
so the ORDER DOES MATTER

In your code, **"Bearer" is just a prefix string used to identify the token type**.
“The token is sent using the Bearer authentication scheme”
the authheader sends the data in the form of bearer and then the token from which we just extract the token and this token is stored in the header

INTERVIEW QUESTION
HOW THE PASSWORD ARE STORED IN THE DATABASE?

Firstly the password is hashed and salt is also added if required and the password is only one way like its converted into random giberish the google employee cannot guess what the password was

so the hashed password is always stored in the db not the actuall password so that is the main thing

and why salting is done to a password?
Because imagine I and bill gates has the same password which means if I know my password I can also assume bill gates password thats why unique salt is always added to a password to make the hash totally different

and along with the password the salt and hash are stored

.put is used to update the user data in db

```
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(req.body, {
        id: req.userId
    })

    res.json({
        message: "Updated successfully"
    })
})
```

updating user data based on the data they send

but to update that they should pass the auth middleware then only they are allowed to edit the data

the zod schema helps to take the input from the user which they want to update

```
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;
```

in this you can see like to find like pallab you type pall it should automatically see you the data

like in sql its like
select * from user like user = “%pall%”;

this is how can you implement like querty in mongodb and multiple query

regex == regular exp

CREATING BANK SCHEMA

first thing is never store float values in the db as js has some issues to fix that we store values and also the number of zeros we multiplied to remove that point values

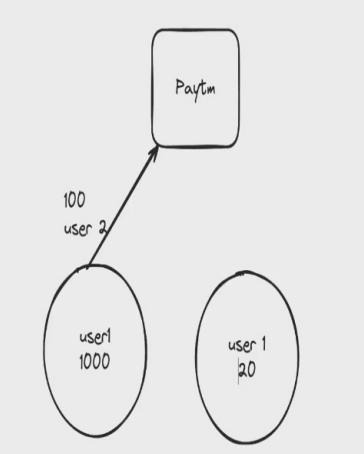
```
const accountSchema = new mongoose.Schema({
    userId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
        required: true
    },
    balance: {
        type: Number,
        required: true
    }
});

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

module.exports = {
    User,
    Account
};
```

also we can add like reference for the db like if there is no user there balance should also not exist so we should create some type of reference between the tables

THE MAIN THING TO LEARN IS TRANSACTIONS

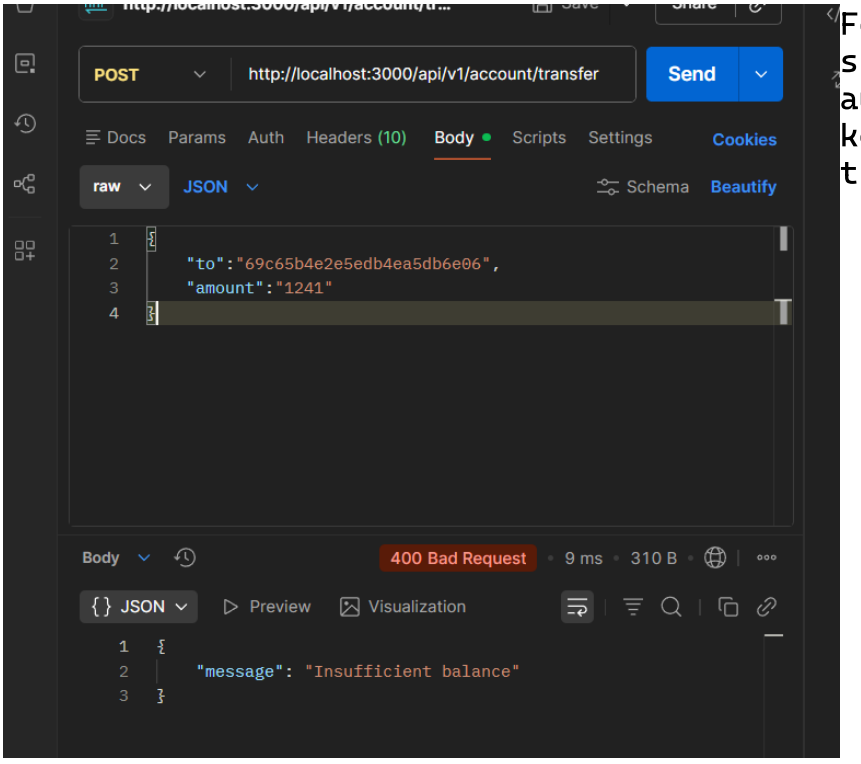


imagine like you sent the data to user 2 and the money is added in the other account but is not deducted from the user account this creates a problem
so to fix that there should exist something like which will let server either to fully complete the process or should revert back to old state don't be in hung state

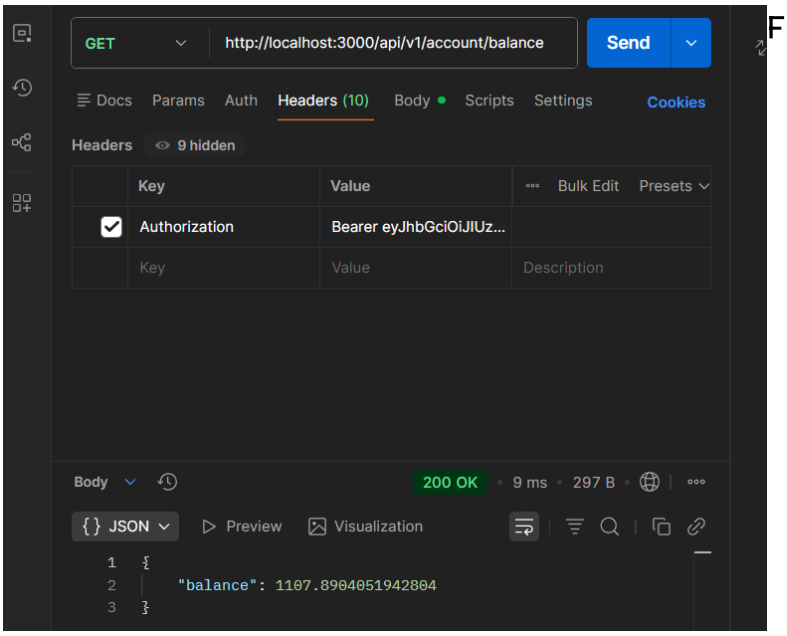
some issue is like when a person just has 100 ruppess and he tries to send money to other two person but at the same time which is not possible tho but then also tries to check mate the system but now you cannot
due to precision errors we never stores the decimal in the db

in eth its like around 17 decimal places of precision so for that also we store the integer only

```
18  router.post("/transfer", authMiddleware, async (req, res) => {
19      const session = await mongoose.startSession();
20      session.startTransaction();
21      const { amount, to } = req.body;
22
23      // Fetch the accounts within the transaction
24      const account = await Account.findOne({ userId: req.userId }).session
        (session);
25      if (!account || account.balance < amount) {
26          await session.abortTransaction();
27          return res.status(400).json({
28              message: "Insufficient balance"
29          });
30      }
31      const toAccount = await Account.findOne({ userId: to }).session(session);
32      if (!toAccount) {
33          await session.abortTransaction();
34          return res.status(400).json({
35              message: "Invalid account"
36          });
37      }
38      // Perform the transfer
39      await Account.updateOne({ userId: req.userId }, { $inc: { balance:
        -amount } }).session(session);
40      await Account.updateOne({ userId: to }, { $inc: { balance: amount } }).
        session(session);
41      // Commit the transaction
42      await session.commitTransaction();
43      res.json({
44          message: "Transfer successful"
45      });
46  });
```



For transferring the money there should the user id in hearders in authorization and should have the key with bearer ____key____ and then in the body



For getting the balance