

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

```
JS index.js JS db.js X
1  const mongoose = require("mongoose");
2  mongoose.connect('mongodb://localhost:27017/paytm')
3  const userSchema = mongoose.Schema({
4    username: String,
5    password: String,
6    firstname: String,
7    lastname: String
8  });
9
10 const User = mongoose.model("User",userSchema);
11 module.exports = {
12   User
13 }
14
```

define a mongoose schema

this is easy way of doing

```
JS index.js JS db.js X
1  const mongoose = require('mongoose');
2  const userSchema = new mongoose.Schema({
3    username: {
4      type: String,
5      required: true,
6      unique: true,
7      trim: true,
8      lowercase: true,
9      minLength: 3,
10     maxLength: 30
11   },
12   password: {
13     type: String,
14     required: true,
15     minLength: 6
16   },
17   firstName: {
18     type: String,
19     required: true,
20     trim: true,
21     maxLength: 50
22   },
23   lastName: {
24     type: String,
25     required: true,
26     trim: true,
27     maxLength: 50
28   }
29 });
30 const User = mongoose.model('User', userSchema);
31 module.exports = {
32   User
33 };
```

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

```
1  const express = require('express');
2
3  const router = express.Router();
4  const zod = require("zod");
5  const { User } = require("../db");
6  const jwt = require("jsonwebtoken");
7  const { JWT_SECRET } = require("../config");
8
9  const signupBody = zod.object({
10   username: zod.string().email(),
11   firstName: zod.string(),
12   lastName: zod.string(),
13   password: zod.string()
14 })
15
16 router.post("/signup", async (req, res) => {
17   const { success } = signupBody.safeParse(req.body)
18   if (!success) {
19     return res.status(411).json({
20       message: "Email already taken / Incorrect inputs"
21     })
22   }
23
24   const existingUser = await User.findOne({
25     username: req.body.username
26   })
```

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 confirming that

```
28   if (existingUser) {
29     return res.status(411).json({
30       message: "Email already taken/Incorrect inputs"
31     })
32   }
33
34   const user = await User.create({
35     username: req.body.username,
36     password: req.body.password,
37     firstName: req.body.firstName,
38     lastName: req.body.lastName,
39   })
40   const userId = user._id;
41
42   const token = jwt.sign({
43     userId
44   }, JWT_SECRET);
45
46   res.json({
47     message: "User created successfully",
48     token: token
49   })
50 }
```

if not found then inserting the data

assigning a jwt and then assigning to the username

```

57 router.post("/signin", async (req, res) => {
58   const { success } = signinBody.safeParse(req.body)
59   if (!success) {
60     return res.status(411).json({
61       message: "Email already taken / Incorrect inputs"
62     })
63   }
64
65   const user = await User.findOne({
66     username: req.body.username,
67     password: req.body.password
68   });
69
70   if (user) {
71     const token = jwt.sign({
72       userId: user._id
73     }, JWT_SECRET);
74
75     res.json({
76       token: token
77     })
78     return;
79   }
80
81
82   res.status(411).json({
83     message: "Error while logging in"
84   })
85 })
86
87 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

```

1 import { user } from './db';
2 const express = require("express");
3 const cors = require("cors");
4
5 app.use(cors());
6 app.use(express.json());
7
8 const mainRouter = require("./routes/index");
9
10 const app = express();
11
12 app.use("/api/v1", mainRouter);
13 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 actual 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

```
94  const updateBody = zod.object({
95    password: zod.string().optional(),
96    firstName: zod.string().optional(),
97    lastName: zod.string().optional(),
98  })
99
100 router.put("/", authMiddleware, async (req, res) => {
101   const { success } = updateBody.safeParse(req.body)
102   if (!success) {
103     res.status(411).json({
104       message: "Error while updating information"
105     })
106   }
107   await User.updateOne(req.body, {
108     id: req.userId
109   })
110
111   res.json({
112     message: "Updated successfully"
113   })
114 })
```

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

```
116 router.get("/bulk", async (req, res) => {
117   const filter = req.query.filter || "";
118
119   const users = await User.find({
120     $or: [{
121       firstName: {
122         "$regex": filter
123       }
124     }, {
125       lastName: {
126         "$regex": filter
127       }
128     }]
129   })
130
131   res.json({
132     user: users.map(user => ({
133       username: user.username,
134       firstName: user.firstName,
135       lastName: user.lastName,
136       _id: user._id
137     })))
138   })
139 })
140
141 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 query in mongodb and multiple query

regex == regular exp

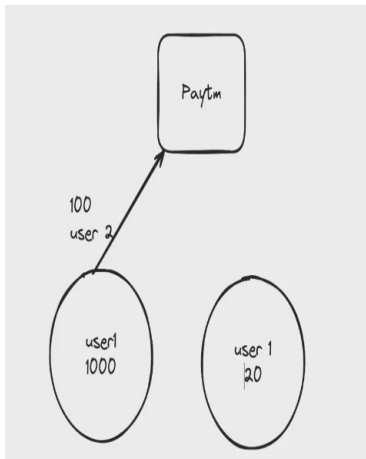
CREATING BANK SCHEMA

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

```
34 const accountSchema = new mongoose.Schema({
35   userId: {
36     type: mongoose.Schema.Types.ObjectId,
37     ref: 'User',
38     required: true
39   },
40   balance: {
41     type: Number,
42     required: true
43   }
44 });
45
46 const Account = mongoose.model('Account', accountSchema);
47 const User = mongoose.model('User', userSchema);
48
49 module.exports = {
50   User,
51   Account
52 };
```

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
25     (session);
26     if (!account || account.balance < amount) {
27         await session.abortTransaction();
28         return res.status(400).json({
29             message: "Insufficient balance"
30         });
31     }
32     const toAccount = await Account.findOne({ userId: to }).session(session);
33     if (!toAccount) {
34         await session.abortTransaction();
35         return res.status(400).json({
36             message: "Invalid account"
37         });
38     }
39     // Perform the transfer
40     await Account.updateOne({ userId: req.userId }, { $inc: { balance:
41     -amount } }).session(session);
42     await Account.updateOne({ userId: to }, { $inc: { balance: amount } }).
43     session(session);
44     // Commit the transaction
45     await session.commitTransaction();
46     res.json({
47         message: "Transfer successful"
48     });
49 });

```

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:3000/api/v1/account/transfer
- Request Body (JSON):**

```

1 {
2   "to": "69c65b4e2e5edb4ea5db6e06",
3   "amount": "1241"
4 }

```
- Response:** 400 Bad Request (9 ms, 310 B)
- Response Body (JSON):**

```

1 {
2   "message": "Insufficient balance"
3 }

```

For transferring the money there should be the user id in headers in authorization and should have the key with bearer `_____key_____` and then in the body

For getting the balance

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:3000/api/v1/account/balance
- Buttons:** Send
- Navigation:** Docs, Params, Auth, Headers (10), Body, Scripts, Settings, Cookies
- Headers:** 9 hidden. One header is visible: Authorization: Bearer eyJhbGciOiJIUz...
- Status:** 200 OK, 9 ms, 297 B
- Body:** JSON format showing the response:

```
1 {  
2   "balance": 1107.8904051942804  
3 }
```