Episode-8

Data Sanitization and Schema Validations

Data Validation :

→ Before adding the incoming data into the DB we need to validate the data. This can be done is many ways. one such way is Schema Validation.

→ In Schema for each field we can add several validators along with the data type like required.

→ required validator → accept a boolean value it tells whether that field is required (or) not.

eg:

```
const userSchema = new mongoose.Schema({
    name: { type: String,
            required: true
    }
});
```

→ we made the name field a mandatory one. If there is no name in the incoming data mongoose will not allow the data to insert into DB rather it throws an error.

→ similarly, there are many validators like unique, minLength, maxLength & so on. that can be used in our Schema to validate the incoming data.

Kindly refer the mongoose documentation → SchemaTypes for more validators

1) How to create a custom validation function ?

→ we can also create our own custom validation by creating a validator function and attach it to a validate property.

→ The validator function returns true or false. we can also provide an error message if the validation fails using message Property.

→ By default, the validator fn runs only when the new document is created and not on update:

→ If we want to run the validation fn on update, we must set the option { runValidators : true } on update API.

eg: User.UpdateOne (id, data, { runValidators : true })

```
//custom validation fn in user Schema.
const userSchema = new Mongoose.Schema({
    age: { type: Number,
           validate: {
               validator: function(value) {
                   return value >= 18 && value <= 65
               },
               message: "age must be above 18 & below 65"
           }
    }
})
```

2) How to add timeStamps in our document?

→ when we want to add timeStamps in our documents we can simply add { timeStamps : true } in our schema as one our field.

→ It will automatically add createdAt and updatedAt fields along with the time.

```
eg: const userschema = new mongoose.Schema({
        name: { type: String },
        age: { type: Number },
    }, { timeStamps : true })
```

Note:

→ The timestamp option has to be second argument to Schema.

API - Level validation:

1) what is API Level Validation?

→ API level validation is a process where data is validated when it is send through an API before being stored to DB.

→ It ensures only correct, complete and secured data is send to DB.

→ It allows only the required fields & filter out unnecessary fields that are injected by any attacks.

→ After the data passes API level validation, it is stored in DB.

→ It can be done along with the Schema - validation for better security.

eg: we are allowing only certain fields to be updated.

```
app.Patch("/user", async (req, res) =>{
    const userId = req.body.userId
    const data = req.body

    try {
      //API Level Validation
      const Allowed_fields = ["about", "PhotoURL", "Skill"]

      const isUpdateAllowed = Object.Keys(data).every((k)=>
          Allowed_fields.includes(k))

      if(!isUpdateAllowed){
          throw new Error("update not allowed")
      }
      const user = await User.findByIdAndUpdate(userId,
                                    data)
      res.send("user updated successfully")
}
```

```
    catch(err){
        res.status(400).("update failed" + err.message)
    }
});
```

Code Explanation:

→ we are creating an "allowed-fields" array which include the fields which can be updated.

→ we are checking the incoming data with the allowed-fields

→ if there is any mismatch in the incoming data like addition of extra field it will throw an Error which is catched by catch block.

→ if there is no Error then data is stored in DB & the document updated successfully.

Data Sanitization:

1) what is data Sanitization?

→ Data sanitization is a process of cleaning (or) altering data to ensure it is free from errors, unwanted content (or) malicious input.

→ It prevents from SQL injection & cross site scripting (XSS) attack.

→ There are libraries like validator.js (or) express-validator makes the validation very easier.

eg: using validator.js

Step1: Installation of validator

npm i validator

Step 2: including it in code for validation

```
//email validation in Schema

const validator = require("validator")

const userSchema = new mongoose.Schema({
    email: { type: String,
             required: true,
             validate: {
                 validator: (value) => {
                     if (!validator.isEmail(value)) {
                         return false;
                     },
                     message: "Invalid Email"
                 }
             }
    })
```

Kindly refer the validator library to know what are the validators available.

---

x