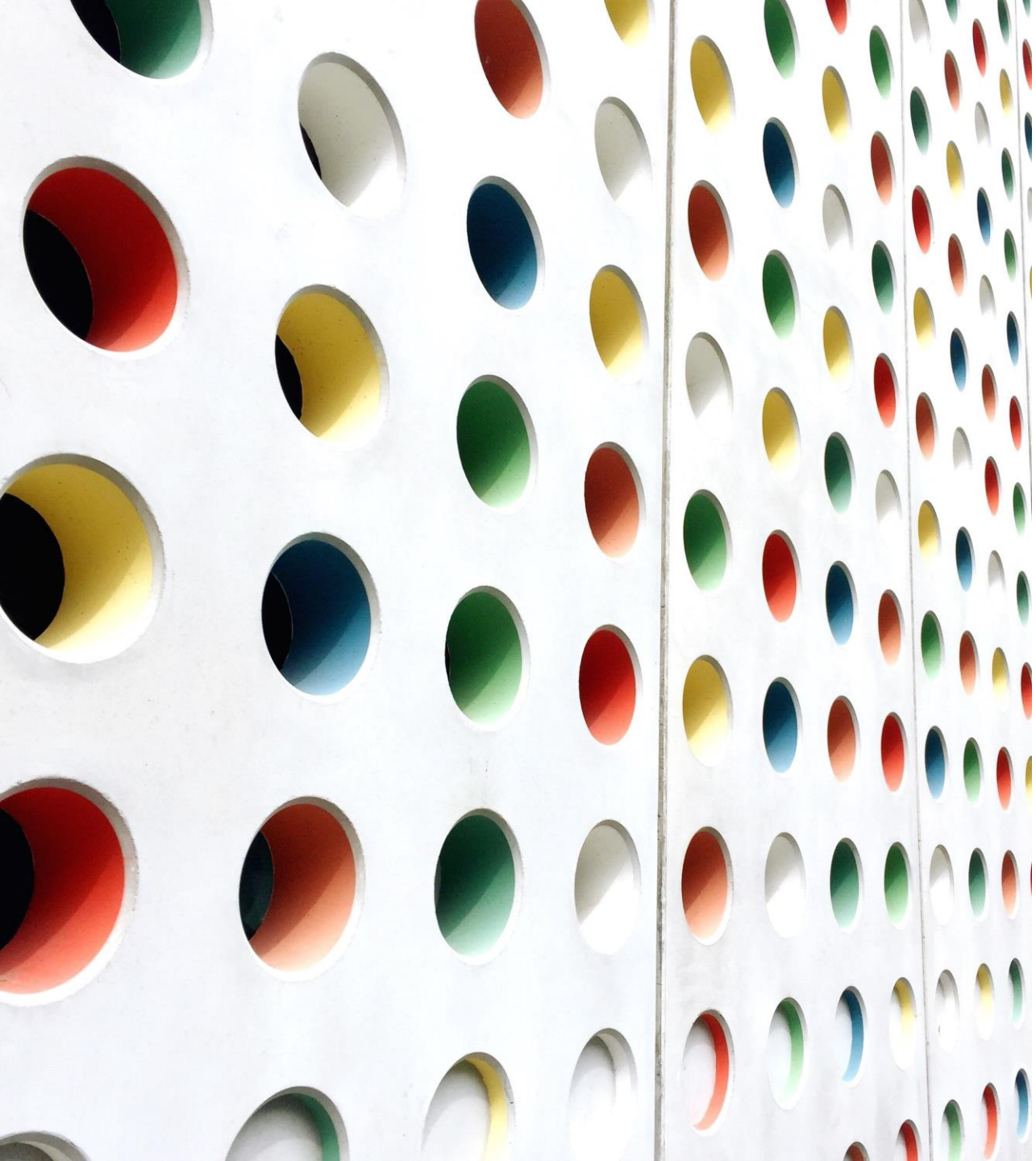


Mongo-Express-Node

Mostly all modern-day web applications have some sort of data storage system at the backend.

For example, if you take the case of a web shopping application, data such as the price of an item would be stored in the database.

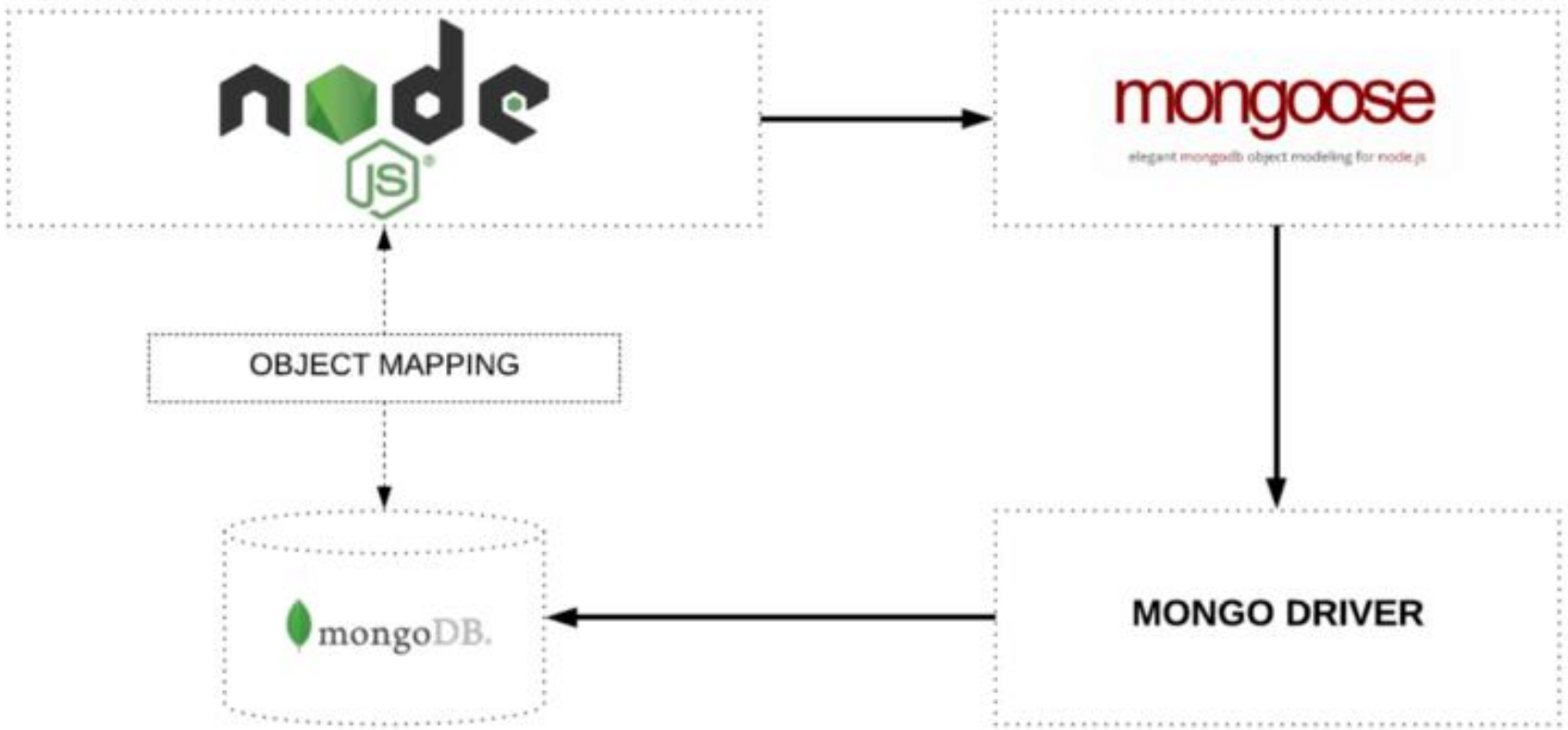
The Node js framework can work with databases with both relational (such as Oracle and MS SQL Server)

and non-relational databases (such as MongoDB).

MongoDB is a NoSQL database. The ability of these databases to store any type of content and particularly in any type of format is what makes these databases so famous.

Mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.



Object Mapping between Node and MongoDB managed via Mongoose

Terminologies

Collections

'Collections' in Mongo are equivalent to tables in relational databases. They can hold multiple JSON documents.

Documents

'Documents' are equivalent to records or rows of data in SQL. While a SQL row can reference data in other tables, Mongo documents usually combine that in a document.

Fields

'Fields' or attributes are similar to columns in a SQL table.

Schema

While Mongo is schema-less, SQL defines a schema via the table definition. A Mongoose ‘schema’ is a document data structure (or shape of the document) that is enforced via the application layer.

Models

‘Models’ are higher-order constructors that take a schema and create an instance of a document equivalent to records in a relational database.

Let’s install Mongoose

```
npm  install  mongoose  --save
```

Using Mongoose

```
var mongoose = require('mongoose');
```

To connect to MongoDB from Node.js using Mongoose package, call connect() function, on the variable referencing to mongoose, with MongoDB Database URI passed as argument to the function.

```
mongoose.connect('mongodb://localhost:27017/database_name');
```

To get a reference to the database specified, use connection() function on Mongoose reference, as shown below :

```
var db = mongoose.connection;
```

Dbconnection.js

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/person');


var db = mongoose.connection;


db.on('error', console.error.bind(console, 'connection error:'));


db.once('open', function() {
  console.log("Connection Successful!");
});
```

Let us consider that we are operating a bookstore and we need to develop a Node.js Application for maintaining the book store. Also we chose MongoDB as the database for storing data regarding books.

```
var BookSchema = mongoose.Schema({
  name: String,
  price: Number,
  quantity: Number
});
```

Compile Schema to Model

Once we derive a custom schema, we could compile it to a model.

Following is an example where we define a model named Book with the help of BookSchema.

```
Const Book=mongoose.model('Book',BookSchema);
```

Initialize a Document

We may now use the model to initialize documents of the Model.

```
var book1 = new Book({
name: ‘Harry Potter’,
price: 800,
quantity: 25
 });
```

Insert Document to MongoDB – To insert a single document to MongoDB, call save() method on document instance.

```
var Book = mongoose.model('Book', BookSchema, 'bookstore');
 var book1 = new Book({ name: 'Introduction to Mongoose', price: 10, quantity: 25 });
 // save model to database
book1.save(function (err, book) {
 if (err) return console.error(err);
 console.log(book.name + " saved to bookstore collection.");
});
```

Conclusion:
Create a folder Models
Create a file book.js

```
const mongoose=require('mongoose')

const bookSchema=new mongoose.Schema({
    name:{
        type:String,
        required:true,
        unique:true
    },
price:{
        type:Number,
        required:true
    },
    Quantity:{
        type:Number,
        required:true
    }
})
module.exports=new mongoose.model('Book',LoginSchema)
```

Create a folder controllers

Bookcontroller.js

```
const express=require('express')
router=express.Router()
const book=require("../models/book")

router.get("/",(req,res,next)=>{
   res.render(“books")
})
router.post("/",async(req,res,next)=>{
   try{
      const b=new book({
         name:req.body.name,
         price:req.body.price,
         quantity:req.body.quantity
})
   const book1=await b.save()
res.send(u)
   }catch(err){
      console.log("You have entered some wrong values")
   }

})
module.exports=router
```

Insert Multiple Documents to MongoDB

To insert Multiple Documents to MongoDB using Mongoose, use Model.collection.insert(docs_array, options, callback_function); method. Callback function has error and inserted_documents as arguments.

```
var books = [{ name: 'Mongoose ', price: 10, quantity: 25 },
             { name: 'NodeJS', price: 15, quantity: 5 },
             { name: 'MongoDB', price: 20, quantity: 2 }];

book.collection.insert(books, function (err, docs) {

  if (err){

    return console.error(err);

  } else {
    console.log("Multiple documents inserted to Collection");
  }
});
```

Fetching data from Mongo Using Node

```
router.get('/:pid',(req,res,next)=>{
const id=req.params.pid
Product.findById(id)
.exec()
.then(doc=>{
   console.log(doc)
   res.status(200).json(doc)
})
.catch(err=>{
   console.log(err)
   res.status(500).json({err:err}) })
})
```

```
router.get('/student',async(req,res)=>{
try{
   const stud=await Student.find()
    console.log(stud)
 res.send(stud)
}
catch(err)
{
   res.send('error'+err)
}
})
```

findOne

```
Product.findOne(name:'Meera')
.exec()
.then(doc=>{
   console.log(doc)
   res.status(200).json(doc)
})
.catch(err=>{
   console.log(err)
   res.status(500).json({err:err}) })
})
```

Deleting data from MongoDB

```
router.delete(‘/book/:name',async(req,res)=>{
    const name=req.params.name;
    try{
    const delbook=await book.deleteOne({name:name})
    res.json(delbook)
}
catch(err)
{
    res.send('error'+err)
}
})
```

Updating MongoDB

```
router.patch("/book/upd",async(req,res)=>{
   try{
   var one={name:"maya"} //existing value
   var two={name:"saya"} //new value
  await Student.updateOne(one,two,(err,collection)=>{
         if(err)
         throw err;
         res.json(collection)
   })    }
catch(err)
{
   res.send('error'+err)
}
})
```

Find by id and update

```
const cid=req.body.id
  login.findByIdAndUpdate({_id:mongoose.Types.ObjectId(cid)},
  {
    username:req.body.username
  },
  function(err, data) {
   if(err){
       res.json({
           status:false,
           message: 'AN ERROR OCCURED'
       })
   }
   else{
       res.json({
           status:true,
           message: 'Category Updated Successfully'
          })
   }
})
```

Find One and Delete

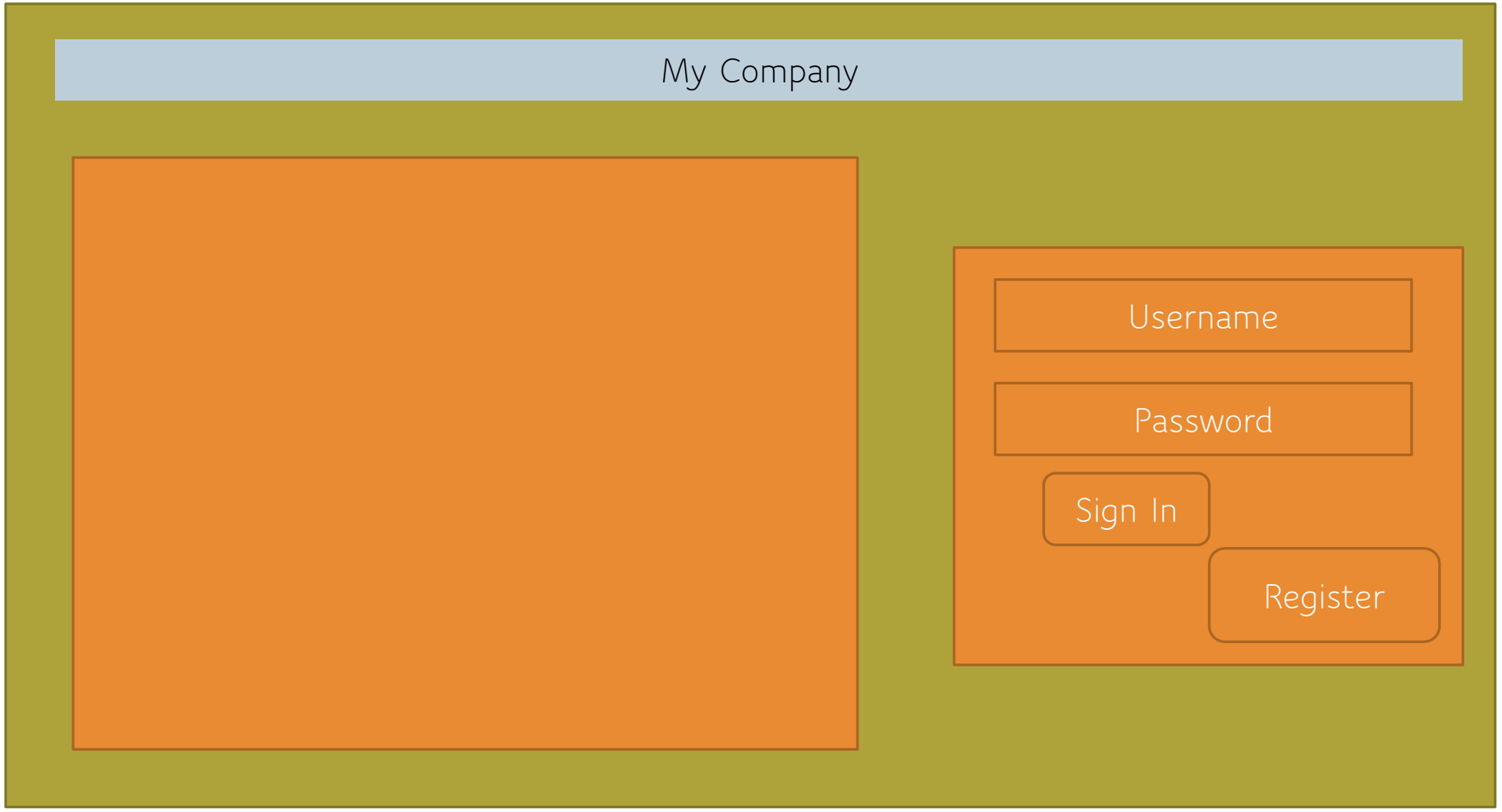
```
router.post('/del',(req,res,next)=>{
        data.findOneAndDelete({name:'Priya'})
         .exec()
         .then(doc=>{
             console.log(doc)
             res.status(200).json({doc})
         })
         .catch(err=>{
             res.status(500).json({
                 error:err
             })
         }) })
```

Employee- Api

Admin
Add an Employee
Update an employee
Delete an Employee
View all employee details

Employee

Registration
View company details
Update self


My Company
Username
Password
Sign In
Register