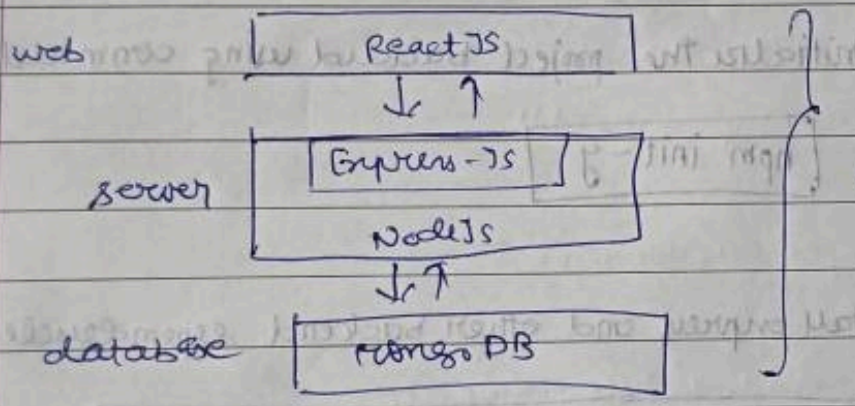
Page No.:
Date:

Mern stack

Mern stack is a javascript stack that is used for easier and faster deployment of full stack web applications.

- when working with MERN stack, developers create implement view layer using react and express and node are used to implement application layer of website then MongoDB is used to implement database layer.



* setup of a Mern stack project

to setup mern stack we need to create a folder structure for both frontend and Backend. then we have to define database schema to store and retrieve data from the database.

Step1 :- creating folder

```
mkdir frontend
mkdir backend
```

Page No.:
Date:

step2:- Navigate to frontend folder
→ cd frontend

step3:- initialize a react project using command
→ `npm create-react-app .` → create in frontend folder only.

step4:- Now navigate to backend folder using
```
cd..
cd backend
```

step5:- Initialize the project backend using command
`npm init -y`

step6:- install express and other backend dependencies
`npm i express mongodb cors dotenv`

So now from this a basic structure of MERN STACK project is created.

☆ After creating index.js file in backend.

```
const express = require('express');
const app = express();
const port = 3000;
```

Page No.:
Date:

```
app.get('/', (req, res) => {
    res.send('Hello from Badaud');
});

app.listen(port, () => {
    console.log(`Server is running on port: ${port}`);
});
```

→ for running project
↳ node index.js

in scripts ↵

```
"start": "node index.js",
"dev": "nodemon index.js"
```

} npm i nodemon

★ MongoDB connectivity

① →
```
import mongoose from 'mongoose';
mongoose.connect(mongo_url).then(() => {
    console.log('App connected to database');
}).catch((error) => {
    console.log(error);
});
```

---

and in above here we can directly call the app.listen after console.log in then block.

Page No.:
Date:

☆ Creating a basic API

creating a basic API in backend involves defining routes, handling HTTP methods, and interacting with a database if necessary. below is a simple example using express, a popular web framework for Node.js.

```
import express from 'express';
const app = express();
const port = 3000;

app.use(express.json());

// sample data for books
const books = [
    { id: 1, title: 'Book1' },
    { id: 2, title: 'book2' },
    { id: 3, title: 'books3' },
];

// creating a api routes for getting all books.

app.get('/api/books', (req, res) => {
    res.json(books);
});
```

⇓

this will get all the books.

Page No.:
Date :

```
// get a single book by id
app.get('/api/books/:id', (req, res) => {
  const book = books.find((b) => b.id === parseInt
              (req.params.id));
  if(!book) return res.status(404).send('book not f');
  res.json(book);
});


// post a new book
app.post('/api/books', (req, res) => {
    cons {title} = req.body;
    if(!title) return res.status(400).send('title');

    const newBook = {
      id: books.length +1,
      title,
    };

    books.push(newBook);
    res.json(newBook);
});


app.listen(port, () => {
   console.log('server is running');
});
```

① DEC-2023 (26-29)

Mern stack tutorial

Basic Book store Project

Page No:
Date:

① in backend →

`npm init -y` → initialize the project added package of json in package.json file and adding all important packages

→ now add a new line in package.json

```
"type": "module",
"main": "index.js",
```

→ `npm i express nodemon` → for starting the server automatically not start again & again (automatically refreshing)

↳ framework

↓ after this

```
"scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js"
},
```

② index.js → starting point of project

```
import {PORT} from "./config.js";
import express from "express";
const/var app = express();

app.listen(PORT, () => {
  console.log("... :${PORT}");
});
```

config.js / keep it in separate file
↳ port

callback fn

server created and its running

↓↓ to run

npm run dev

or

npm run start

Go to developer tools
↳ network
→ to check for request

Page No.:
Date:

→ to get a request / send

```
app.get('/', (request, response) => {
    response.send('Hello world!');
});
```

or

```
return response.status(234).send('');
});
```

---

③ MongoDB

password → ar.

```
export const mongoDBURL = '                    ';
```

→

```
import mongoose from 'mongoose';
mongoose.connect(mw).then(() => {
    console.log("App connected");
    app.listen(
    ,
}).catch((error) => {
    console.log(error);
});
```

---

example of model in mongoose

```
import mongoose from "mongoose";

export const Book = mongoose.model('Book', bookSchema);
                    ↑ name

const bookSchema = mongoose.Schema({
    {
    },
    {},
});
```

Page No.:
Date:

for example : Complete

```
import express from "express"
import {PORT, mongoDBURL} from './config.js';
import mongoose from "mongoose";
import cors from 'cors';
import booksRoute from './routes/booksRoutes.js';

const app = express();        → main app.
```

→ (1)

```
app.use(express.json());  → middleware for parsing request body
app.use(cors());
```

→ (2)

```
app.get('/', (request, response) => {        → get request
    console.log(request);
    return response.status(234).send('    ');
});
```

→ (3)

```
app.use('/books', booksRoute);
```

for routing purpose → (4)

```
mongoose.connect(mongoDBURL).then(() => {
    console.log('App connected to database');
    app.listen(PORT, () => {
        console.log(`App is running : ${PORT}`);
    });
})
.catch((error) => {
    console.log(error);
});
```

→ (5)

mongodb connectivity

Page No.:
Date:

Routes part detailed explanation

```
import express from "express"
import {Book} from '../models/ bookModel.js';
const router = express.Router();
```

⟹ importing the model into route & creating a router.

① router.post ('/', async (request, response) => {
  try {
    if (
      !request.body.title || !request.body.author ||
      !request.body.publishYear)
    { return response.status(400).send ({
      message: 'Send all required fields: title, author, publishYear',
    });
    }
    const newBook = {
      title: request.body.title,
      author: request.body.author,
      publishYear: request.body.publishYear,
    };
    const book = await Book.create(newBook);
    return response.status(201).send(book);
  } catch (error) {
    console.log(error.message);
    response.status(500).send({ message: error.message });
  }
});

if not written 400 status display

if fields are present new book objects will be created

Imp creating

otherwise error

② Get all books from database

router.get('/', async (request, response) => {
  try {
    const books = await Book.find({});
    return response.status(200).json({
      count: books.length,
      data: books,
    });
  } catch (error) {
    console.log(error.message);
    response.status(500).send({ message: error.message });
  }
});

return an array containing data

otherwise error

③ Get one book from database by id

router.get('/:id', async (request, response) => {
  try {
    const { id } = request.params;
    const book = await Book.findById(id);
    return response.status(200).json(book);
  } catch (error) {
    console.log(error.message);
    response.status(500).send({ message: error.message });
  }
});

Book.findById(id);

Book.findById(id);

```
Book.create();
Book.find({});
Book.findByIdAndUpdate();
Book.findByIdAndDelete();
```

id, request.body

Page No.:
Date:

② Get all

```
router.get('/', async (request, response) => {
  try {
    const book = await Book.find({});
    response.status(200).json({
      count: books.length,
      data: books,
    });
  } catch (error) {
    console.log(error.message);
    response.status(500).send({ message: error.message });
  }
});
```

for getting all

return an array containing data

otherwise error

③ Get book by ID

```
router.get('/', async (request, response) => {
  try {
    const { id } = request.params;
    const book = await Book.findById(id);
    return response.status(200).json(book);
  } catch (error) {
    console.log(error.message);
    response.status(500).send({ message: error.message });
  }
});
```

by id

Page No.
Date :

④ edit / update

```
router.put('/:id', async (request, response) => {
  try {
    if(!request.body.title || !request.body.author ||
    !request.body.publishYear) {
      return response.status(400).send({
      message: })}
  }
```

imp

```
const {id} = request.params;
const result = await Book.findByIdAndUpdate
                (id, request.body);
```

```
if (!result) {
  return response.status(400).json({ mes: ' ' });
  return response.status(200).send({ mes = })
} catch (error) {
  console.log(error.message);
  response.status(500).send({ ; });
}
});
```

⑤ Delete

```
router.delete('/', async (request, response) => {
  try {
    const {id} = request.params;
    const result = await Book.findByIdAndDelete(id);
```

imp

Page No.:
Date:

Model :-

```
import mongoose from 'mongoose';
const bookSchema = mongoose.Schema({
  title: {
      type: String,
      required: true,
  },
  author: {
      type: String,
      required: true,
  },
  publishYear: {
      type: Number,
      required: true,
  },
  releasedDate: {
      type: Date,
      required: true,
  },
  coverImage: {
      type: String,
      required: true,
  },
  publicationCalender: {
      type: Date,
  },
  isBestSeller: {
      type: Boolean,
      default: false,
  },
},
{ timestamps: true, });
```

Syn: of model

```
export const Book = mongoose.model('Book', bookSchema);
```