ONLINE LEARNING PLATFORM

USING MERN

PROJECT REPORT

TEAM MEMBERS:

1. **NARENDRAN K**
2. **MUKESH VARMA P**
3. **MONISHA R**
4. **REKHA J**

GITHUB REPO :

https://github.com/iamnarendran/ONLINE-LEARNING-PLATFORM-USING-MERN.git

ABSTRACT

This project report presents an innovative online learning platform that leverages the MERN stack (MongoDB, Express.js, React.js, and Node.js) to enhance the educational experience. Our goal is to create a dynamic and interactive platform that facilitates seamless learning, knowledge sharing, and collaboration among students and instructors. The report provides an overview of the platform's architecture, covering the front-end and back-end development, database integration, and deployment process. It further discusses the development process, implementation setup, testing procedures, and results. This comprehensive documentation showcases how modern full-stack technologies can be utilized effectively to create a robust and engaging online learning platform, meeting the evolving needs of students, educators, and educational institutions.

INTRODUCTION

In the realm of online learning platforms, the demand for dynamic, responsive, and high-performing platforms that cater to the needs of learners and educators alike has soared. To meet these demands, the MERN stack—a combination of MongoDB, Express.js, React.js, and Node.js—has become a compelling choice for building engaging and scalable online learning environments. This project focuses on developing a full-stack online learning platform that leverages the MERN stack to provide an intuitive user interface, efficient back-end processing, and reliable data management.

This project showcases how the integration of these technologies offers flexibility, rapid development, and scalability. The platform is designed to cater to [target audience, e.g., students, teachers, administrators] and includes features such as [list key features or functionalities, e.g., user authentication, course management, video streaming, etc.].

PROJECT OBJECTIVES

The main objectives of this project are:
1. Develop a full-stack web application using the MERN stack that is scalable, maintainable, and responsive.

2. Implement a user-friendly front-end interface that allows intuitive interaction for users.

3. Build a robust back-end system capable of handling multiple API requests, ensuring data integrity and fast response times.

4. Design a NoSQL database with MongoDB to store, retrieve, and manage data efficiently.

5. Ensure cross-platform functionality by optimizing the application for various devices (desktop, mobile).

6. Perform comprehensive testing of the system to ensure functionality, security, and reliability.

TECHNOLOGY STACK

The project is developed using the MERN stack, which consists of:

MongoDB: A NoSQL database that uses JSON-like documents to store data, ensuring high flexibility and scalability.- Express.js: A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

React.js: A JavaScript library for building user interfaces, primarily used to develop the front end of the application with a component-based architecture.
Node.js: A JavaScript runtime built on Chrome's V8 JavaScript engine, used to build the back-end services and APIs.

SYSTEM ARCHITECTURE

Front-end Architecture (React.js)

The front-end is built using React.js, following a component-based architecture. React allows the creation of reusable components, resulting in a more organized and scalable codebase. The key elements of the front-end architecture include:

UI Components: Each functional aspect of the UI is encapsulated into components (e.g., forms, buttons, data tables).

State Management: React hooks, such as useState and useEffect, are used to manage the application's state.

Routing: React Router is used to handle multiple routes, ensuring that the application can support various pages and views seamlessly.

Back-end Architecture (Node.js, Express.js)

The back-end is structured to handle all server-side logic, processing client requests and interacting with the MongoDB database.
The architecture consists of:

API Endpoints: The server exposes RESTful API endpoints using Express.js to handle data requests (e.g., fetching, updating, and deleting data).

Authentication: The back-end incorporates user authentication mechanisms (e.g., JWT, OAuth) to ensure secure access to resources.

Middleware: Express.js middleware is used for tasks such as request parsing, logging, and handling errors.

Database (MongoDB) MongoDB is as the database system for this project. The database is designed to store user data, application data,

and other necessary information in a flexible and scalable manner.The document-oriented nature of MongoDB allows for faster queries and easier data manipulation compared to relational databases.

Collections and Documents: The database is structured into collections, with each collection containing documents that represent individual records.

Schema Design: Although MongoDB is schema-less, this project adopts a semi-structured schema to enforce some level of organization within the data. Key collections include [list key collections, e.g., users, transactions, products, etc.].

Data Security: User data is secured through encryption, and sensitive information such as passwords is hashed using algorithms like bcrypt.

IMPLEMENTATION

The project setup involves the installation and configuration of the MERN stack components:

1. Install MongoDB: Set up a MongoDB instance locally or using a cloud service such as MongoDB Atlas.

2. Node.js and Express.js Setup: Initialize the Node.js application, install necessary dependencies (Express.js, Mongoose, etc.), and configure the server.

3. React.js Setup: Use Create React App to scaffold the front-end, installing necessary packages such as React Router and Redux.

Development Process

The development process followed an Agile methodology with iterative sprints. Key phases included:

1. Requirements Analysis: Defined the application’s features, user roles, and system functionalities.

2. Frontend Development: Developed individual React components, integrated APIs, and implemented responsive design techniques.

3. Backend Development: Developed RESTful APIs, implemented database interactions, and managed authentication.

4. Testing and Debugging: Conducted unit testing for React components and back-end services, as well as integration testing to ensure smooth interaction between all system components.

Testing

Testing is a critical component of the implementation process. The testing methodology included:

Unit Testing: Using tools such as Jest and Mocha to test individual components and back-end functions.

Integration Testing: Ensured that the front-end and back-end systems work together as expected.

End-to-End Testing: Automated tests using Cypress or Selenium to simulate user interaction and ensure that the entire application functions as expected.

Performance Testing: Tools like Apache JMeter were used to evaluate the system's load capacity and response time under various conditions.

PROJECT STRUCTURE

```
ONLINE-LEARNING-PLATFORM-USING-MERN/
├── backend/
│   ├── models/
│   │   ├── Addcourse.js
│   │   ├── Addstudents.js
│   │   ├── Newsandannu.js
│   │   ├── Request.js
│   │   └── userlist.js
│   ├── routes/
│   │   ├── Addcourse.js
│   │   ├── Addstudents.js
│   │   ├── Newsandannu.js
│   │   ├── Request.js
│   │   └── userlist.js
│   ├── config/
│   │   └── db.js
├── frontend/
│   ├── public/
│   ├── src/
│   │   ├── pages/
│   │   ├── app.js
│   │   └── index.js
│   ├── build/
│   │   ├── static/
│   │   │   ├── css
│   │   │   │   └── main.css
│   │   │   ├── js
│   │   │   │   └── main.js
│   │   │   ├── media
└── README.md
```

1. Backend Files

`Addcourse.js` (Models file)

```
const mongoose = require("mongoose")

const AddcourseSchema = mongoose.Schema({
   courseName: String,
})

module.exports = mongoose.model("Addcourse",
AddcourseSchema);
```

`Addstudents.js` (Models file)

```
const mongoose = require("mongoose")

const AddstudentsSchema = mongoose.Schema({
   name: String,
   email: String,
   password: String,
   grade: String,
   areaOfStudy: String,
   skills: Array,
   language: String,
   qualification: String,
   specialization: String,
   teachingExp: String,
   type: String,
   addData: String,
   timing: Array,
   videoLink: String,
   profilePhoto: String,
```

```
})

module.exports = mongoose.model("Addstudents",
AddstudentsSchema);
```

`Newsandannu.js` (Models file)

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

const userSchema = mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
}, {
  timestamps: true,
});

const User = mongoose.model('User', userSchema);
module.exports = User;
```

`Request.js` (Models file)

```
const mongoose = require("mongoose")

const RequestSchema = mongoose.Schema({
```

```
   to: String,
   from: String,
   status: String,
   mode: String,
   ambience: String,
   course: String,
})

module.exports = mongoose.model("Request",
RequestSchema);
```

`userlist.js` (Models file)

```
const mongoose = require("mongoose")

const UserlistSchema = mongoose.Schema({
   firstname:String,
   lastname:String,
   email:String,
   password:String,
   type:String,
})

module.exports = mongoose.model("Userlist",
UserlistSchema);
```

`Addcourse.js` (Route file)

```
const express = require("express");
const router = express.Router();
const Addcourse = require("../models/Addcourse");
const redis= require('redis');
const util = require('util');
```

```
const redisUrl = "redis://127.0.0.1:6379"
const client = redis.createClient(redisUrl)

client.set=util.promisify(client.set)
client.get=util.promisify(client.get)

router.get("/", async (req, res) => {
  try {
    const addcourse = await Addcourse.find();
    res.status(200).json({
      data: addcourse,
    });
  } catch (error) {
    res.status(500).json({
      message: "Server error",
    });
  }
});

router.get("/:id", async (req, res) => {
  try {
//Redis-Caching
const courseId = req.params.id;
    const cachedCourse= await
client.get(`course-${courseId}`)
  if(cachedCourse){
    const course = JSON.parse(cachedCourse);
    return res.status(200).json({
      data: course,
    });
  }
  const course = await Addcourse.findById(courseId);
  if (!course) {
    return res.status(404).json({
      message: "Course not found",
```

```
    });
  }

  await client.set(`course-${courseId}`,
JSON.stringify(course));

  //over
   res.status(200).json({
      data: addcourse,
    });
  } catch (error) {
    res.status(500).json({
      message: "Server error",
    });
  }
});

router.post("/", async (req, res) => {
  try {
    const addcourse = new Addcourse(req.body);
    const newaddcourse = await addcourse.save();
    res.status(200).json({
      data: newaddcourse,
    });
  } catch (err) {
    console.log(err);
    res.status(500).send("Server error");
  }
});


router.post("/login",async (req, res)=>{
  try{
    const newUser=await
Addcourse.findOne({email:req.body.email})
    if(newUser){
```

```
      if(newUser.password===req.body.password){
        res.status(200).json({
          msg:"ok",
          data:newUser,
        })

      }else{
        res.status(200).json({
          msg:"inccorect password",
        })
      }
    }else{
      res.status(200).json({
        msg:"invalid user"
      })
    }
  }catch (err){
    console.log(err)
    res.status(500).send("server error")
  }
})

router.put("/:id", async (req, res) => {
  try {
    const addcourse = await
Addcourse.findById(req.params.id);

    if (!addcourse) {
      return res.status(400).json({ message: "Addcourse
does not exist" });
    }
    addcourse.email = req.body.email || addcourse.email;
    addcourse.password = req.body.password ||
addcourse.password;
    const updatedAddcourse = await addcourse.save();
```

```
    res.status(200).json({
      data: updatedAddcourse,
    });
  } catch (error) {
    console.log(error.message);
    res.status(500).json({ message: "Server error" });
  }
});

router.delete("/:id", async (req, res) => {
  try {
    await Addcourse.findByIdAndRemove(req.params.id);

    res.status(200).json({
      message: "Addcourse is deleted",
    });
  } catch (error) {
    console.error(error);
    res.status(500).send("Server error");
  }
});

module.exports = router;
```

`Addstudents.js` (Route file)

```
const express = require("express");
const router = express.Router();
const Addstudents = require("../models/Addstudents");

router.get("/", async (req, res) => {
  try {
    const addstudents = await Addstudents.find();
    res.status(200).json({
      data: addstudents,
```

```
    });
  } catch (error) {
    res.status(500).json({
      message: "Server error",
    });
  }
});

router.get("/:id", async (req, res) => {
  try {
    const addstudents = await
Addstudents.findById(req.params.id);

    res.status(200).json({
      data: addstudents,
    });
  } catch (error) {
    res.status(500).json({
      message: "Server error",
    });
  }
});

router.post("/", async (req, res) => {
  try {
    const addstudents = new Addstudents(req.body);
    const newaddstudents = await addstudents.save();
    res.status(200).json({
      data: newaddstudents,
    });
  } catch (err) {
    console.log(err);
    res.status(500).send("Server error");
  }
});
```

```
router.post("/login", async (req, res) => {
  try {
    const newUser = await Addstudents.findOne({ email:
req.body.email })
    if (newUser) {
      if (newUser.password === req.body.password) {
        res.status(200).json({
          msg: "ok",
          data: newUser,
        })

      } else {
        res.status(200).json({
          msg: "inccorect password",
        })
      }
    } else {
      res.status(200).json({
        msg: "invalid user"
      })
    }
  } catch (err) {
    console.log(err)
    res.status(500).send("server error")
  }
})

router.put("/:id", async (req, res) => {
  try {
    const addstudents = await
Addstudents.findById(req.params.id);
    if (!addstudents) {
      return res.status(400).json({ message: "Addstudents
does not exist" });
    }
```

```
    addstudents.name = req.body.name || addstudents.name;
    addstudents.email = req.body.email ||
addstudents.email;
    addstudents.password = req.body.password ||
addstudents.password;
    addstudents.grade = req.body.grade ||
addstudents.grade;
    addstudents.areaOfStudy = req.body.areaOfStudy ||
addstudents.areaOfStudy;
    addstudents.skills = req.body.skills ||
addstudents.skills;
    addstudents.language = req.body.language ||
addstudents.language;
    addstudents.qualification = req.body.qualification ||
addstudents.qualification;
    addstudents.specialization = req.body.specialization
|| addstudents.specialization;
    addstudents.teachingExp = req.body.teachingExp ||
addstudents.teachingExp;
    addstudents.type = req.body.type || addstudents.type;
    addstudents.addData = req.body.addData ||
addstudents.addData;
    addstudents.timing = req.body.timing ||
addstudents.timing;
    addstudents.videoLink = req.body.videoLink ||
addstudents.videoLink;
    addstudents.profilePhoto = req.body.profilePhoto ||
addstudents.profilePhoto;
    const updatedAddstudent = await addstudents.save();

    res.status(200).json({
      data: updatedAddstudent,
    });
  } catch (error) {
    console.log(error.message);
    res.status(500).json({ message: "Server error" });
```

```
  }
});

router.delete("/:id", async (req, res) => {
  try {
    await Addstudents.findByIdAndRemove(req.params.id);

    res.status(200).json({
      message: "Addstudents is deleted",
    });
  } catch (error) {
    console.error(error);
    res.status(500).send("Server error");
  }
});

module.exports = router;
```

`Newsandannu.js` (Models file)

```
const express = require("express");
const router = express.Router();
const Newsandannu = require("../models/Newsandannu");

router.get("/", async (req, res) => {
  try {
    const newsandannu = await Newsandannu.find();
    res.status(200).json({
      data: newsandannu,
    });
  } catch (error) {
    res.status(500).json({
      message: "Server error",
    });
  }
```

```
});

router.get("/:id", async (req, res) => {
  try {
    const newsandannu = await
Newsandannu.findById(req.params.id);

    res.status(200).json({
      data: newsandannu,
    });
  } catch (error) {
    res.status(500).json({
      message: "Server error",
    });
  }
});

router.post("/", async (req, res) => {
  try {
    const newsandannu = new Newsandannu(req.body);
    const newnewsandannu = await newsandannu.save();
    res.status(200).json({
      data: newnewsandannu,
    });
  } catch (err) {
    console.log(err);
    res.status(500).send("Server error");
  }
});


router.post("/login",async (req, res)=>{
  try{
    const newUser=await
Newsandannu.findOne({email:req.body.email})
    if(newUser){
```

```
      if(newUser.password===req.body.password){
        res.status(200).json({
          msg:"ok",
          data:newUser,
        })

      }else{
        res.status(200).json({
          msg:"inccorect password",
        })
      }
    }else{
      res.status(200).json({
        msg:"invalid user"
      })
    }
  }catch (err){
    console.log(err)
    res.status(500).send("server error")
  }
})

router.put("/:id", async (req, res) => {
  try {
    const newsandannu = await
Newsandannu.findById(req.params.id);

    if (!newsandannu) {
      return res.status(400).json({ message: "Newsandannu
does not exist" });
    }
    newsandannu.email = req.body.email ||
newsandannu.email;
    newsandannu.password = req.body.password ||
newsandannu.password;
    const updatedNewsandannu = await newsandannu.save();
```

```
    res.status(200).json({
      data: updatedNewsandannu,
    });
  } catch (error) {
    console.log(error.message);
    res.status(500).json({ message: "Server error" });
  }
});

router.delete("/:id", async (req, res) => {
  try {
    await Newsandannu.findByIdAndRemove(req.params.id);

    res.status(200).json({
      message: "Newsandannu is deleted",
    });
  } catch (error) {
    console.error(error);
    res.status(500).send("Server error");
  }
});

module.exports = router;
```

`request.js` (Route file)

```
const express = require("express");
const router = express.Router();
const Request = require("../models/Request");

router.get("/", async (req, res) => {
  try {
    const request = await Request.find();
    res.status(200).json({
```

```
      data: request,
    });
  } catch (error) {
    res.status(500).json({
      message: "Server error",
    });
  }
});

router.get("/:id", async (req, res) => {
  try {
    const request = await
Request.findById(req.params.id);

    res.status(200).json({
      data: request,
    });
  } catch (error) {
    res.status(500).json({
      message: "Server error",
    });
  }
});

router.post("/", async (req, res) => {
  try {
    const request = new Request(req.body);
    const newrequest = await request.save();
    res.status(200).json({
      data: newrequest,
    });
  } catch (err) {
    console.log(err);
    res.status(500).send("Server error");
  }
});
```

```
router.post("/login", async (req, res) => {
  try {
    const newUser = await Request.findOne({ email:
req.body.email })
    if (newUser) {
      if (newUser.password === req.body.password) {
        res.status(200).json({
          msg: "ok",
          data: newUser,
        })

      } else {
        res.status(200).json({
          msg: "inccorect password",
        })
      }
    } else {
      res.status(200).json({
        msg: "invalid user"
      })
    }
  } catch (err) {
    console.log(err)
    res.status(500).send("server error")
  }
})

router.put("/:id", async (req, res) => {
  try {
    const request = await
Request.findById(req.params.id);

    if (!request) {
      return res.status(400).json({ message: "Request
```

```
does not exist" });
    }
    request.to = req.body.to || request.to;
    request.from = req.body.from || request.from;
    request.status = req.body.status || request.status;
    request.mode = req.body.mode || request.mode;
    request.ambience = req.body.ambience ||
request.ambience;
    request.course = req.body.course || request.course;
    const updatedRequest = await request.save();

    res.status(200).json({
      data: updatedRequest,
    });
  } catch (error) {
    console.log(error.message);
    res.status(500).json({ message: "Server error" });
  }
});

router.delete("/:id", async (req, res) => {
  try {
    await Request.findByIdAndRemove(req.params.id);

    res.status(200).json({
      message: "Request is deleted",
    });
  } catch (error) {
    console.error(error);
    res.status(500).send("Server error");
  }
});

module.exports = router;
```

`userlist.js` (Route file)

```
const express = require("express");
const router = express.Router();
const Userlist = require("../models/Userlist");

router.get("/", async (req, res) => {
  try {
    const userlist = await Userlist.find();
    res.status(200).json({
      data: userlist,
    });
  } catch (error) {
    res.status(500).json({
      message: "Server error",
    });
  }
});

router.get("/:id", async (req, res) => {
  try {
    const userlist = await
Userlist.findById(req.params.id);

    res.status(200).json({
      data: userlist,
    });
  } catch (error) {
    res.status(500).json({
      message: "Server error",
    });
  }
});

router.post("/", async (req, res) => {
  try {
```

```
    const userlist = new Userlist(req.body);
    const newuserlist = await userlist.save();
    res.status(200).json({
      data: newuserlist,
    });
  } catch (err) {
    console.log(err);
    res.status(500).send("Server error");
  }
});


router.post("/login",async (req, res)=>{
  try{
    const newUser=await
Userlist.findOne({email:req.body.email})
    if(newUser){
      if(newUser.password===req.body.password){
        res.status(200).json({
          msg:"ok",
          data:newUser,
        })

      }else{
        res.status(200).json({
          msg:"inccorect password",
        })
      }
    }else{
      res.status(200).json({
        msg:"invalid user"
      })
    }
  }catch (err){
    console.log(err)
    res.status(500).send("server error")
```

```
  }
})

router.put("/:id", async (req, res) => {
  try {
    const userlist = await
Userlist.findById(req.params.id);

    if (!userlist) {
      return res.status(400).json({ message: "Userlist
does not exist" });
    }
     userlist.lastname = req.body.lastname ||
userlist.lastname;
    userlist.email = req.body.email || userlist.email;
    userlist.password = req.body.password ||
userlist.password;
    const updatedUserlist = await userlist.save();

    res.status(200).json({
      data: updatedUserlist,
    });
  } catch (error) {
    console.log(error.message);
    res.status(500).json({ message: "Server error" });
  }
});

router.delete("/:id", async (req, res) => {
  try {
    await Userlist.findByIdAndRemove(req.params.id);

    res.status(200).json({
      message: "Userlist is deleted",
    });
  } catch (error) {
```

```
    console.error(error);
    res.status(500).send("Server error");
  }
});

module.exports = router;
```

Db.js (Config File)

```
const mongoose = require("mongoose");

const connectDB = () => {
    mongoose
        .connect(process.env.mongoURI, {
            useNewUrlParser: true,
            useUnifiedTopology: true,
        })
        .then(() => console.log("MongoDB Connected"))
        .catch((err) => {
            console.error(err.message);
            process.exit(1);
        });
};

module.exports = connectDB;
```

2. Frontend Files

`app.css` (src \ pages file)

```
.App {
  text-align: center;
}
```

```css
.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

`App.js` (src \ pages file)

```
import React from 'react'
import {HashRouter,Routes,Route} from 'react-router-dom'
import Login from './Pages/Login'
import Sidebar from './Pages/Sidebar'
import Signupteacher from './Pages/Signupteacher'
import Header from './Pages/Header'
import Addcts from './Pages/Addcts'
import Newsandannc from './Pages/Newsandannc'
import Allcourses from './Pages/Allcourses'
import Upcomeing from './Pages/Upcomeing'
import Viewcourses from './Pages/Viewcourses'
import Signupstudent from './Pages/Signupstudent'
import Setting from './Pages/Setting'
import Students from './Pages/Students'

function App() {
  return (
    <HashRouter>
      <Routes>
        <Route path='/' element={<Login/>}></Route>
        <Route path='/Signupteacher'
element={<Signupteacher/>}></Route>
        <Route path='/Signupstudent'
element={<Signupstudent/>}></Route>
        <Route path='/Sidebar'
element={<Sidebar/>}></Route>
        <Route path='/Header'
element={<Header/>}></Route>
        <Route path='/Addcts'
element={<Addcts/>}></Route>
        <Route path='/Newsandannc'
element={<Newsandannc/>}></Route>
        <Route path='/Allcourses'
element={<Allcourses/>}></Route>
```

```
        <Route path='/Upcomeing'
element={<Upcomeing/>}></Route>
        <Route path='/Viewcourses'
element={<Viewcourses/>}></Route>
        <Route path='/Setting'
element={<Setting/>}></Route>
        <Route path='/Students'
element={<Students/>}></Route>
      </Routes>
    </HashRouter>
  )
}

export default App
```

`index.js` (src \ pages file)

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your
app, pass a function
// to log results (for example:
reportWebVitals(console.log))
```

```
// or send to an analytics endpoint. Learn more:
https://bit.ly/CRA-vitals
reportWebVitals();
```

`index.css` (src \ pages file)

```
body{
  margin: 0px;
  padding: 0px;
}

.loginmain{
  background-image: url("loginbg.jpg");
  background-repeat: no-repeat;
  background-size: cover;
}
.signupbg{
  background-image:
url("https://cdn.pixabay.com/photo/2017/08/02/01/11/peopl
e-2569404_1280.jpg");
  background-repeat: no-repeat;
  background-size: cover;
  background-position: right;
}

.sideblack{
  box-shadow: rgba(0, 0, 0, 0.25) 0px 54px 55px, rgba(0,
0, 0, 0.12) 0px -12px 30px, rgba(0, 0, 0, 0.12) 0px 4px
6px, rgba(0, 0, 0, 0.17) 0px 12px 13px, rgba(0, 0, 0,
0.09) 0px -3px 5px;
}

::-webkit-scrollbar {
  width: 5px;
```

```
}
::-webkit-scrollbar-track {
  background: #f1f1f1;
  box-shadow: inset 0 0 5px grey;
  border-radius: 10px;
}
::-webkit-scrollbar-thumb {
  background: black;
  border-radius: 10px;
}



.res-sidebar{
  display: none !important;
}



@media screen and (max-width: 767px) {
.lgnd1{
  width:100% !important;
}

.head{
  /* display: none !important; */
  flex-direction: column !important;
 margin-top: 5% !important;
}
.head-info{
  width: 90% !important;
}
.head-info2{
  width: 90% !important;
}
.head-info1{
  margin-top: 5% !important;
  margin-bottom: 5% !important;
```

```
  width: 90% !important;
  display: flex !important;
  justify-content: space-between !important;
}
.sideblack{
  display: none !important;
}
.res-sidebar{
  display: flex !important;
  align-items: center !important;
  justify-content: space-evenly !important;
}
.all-courses{
  width: 100% !important;
  margin-left: 0% !important;
  height: 300px !important;
}
.all-courses-1{
  margin-top: 15% !important;
}
.all-course-side{
  display: none !important;
}
.upcoming-res{
  display: none !important;
}
.upcoming-res1{
  width:70% !important;
}
.set-res1{
  flex-direction: column !important;
}
.headres{
  left: 0% !important;
}
.noti-head{
```

```
  flex-direction: column !important;

}
.btn-noti{
  width: 60% !important;
}
.all-course-div{
  flex-direction: column !important;
  height: auto !important;
}
.all-course-div1{
  width: 100% !important;
  height: auto !important;
}

}
```

`main.css` (build \ static file)

```
body {
    margin: 0;
    padding: 0;
}

.loginmain {
    background-image:
url(/static/media/loginbg.e2e019d3b4b6c2cc55a7.jpg);
}

.loginmain,
.signupbg {
    background-repeat: no-repeat;
    background-size: cover;
}
```

```css
.signupbg {
    background-image:
url(https://cdn.pixabay.com/photo/2017/08/02/01/11/people
-2569404_1280.jpg);
    background-position: 100%;
}

.sideblack {
    box-shadow:
        0 54px 55px rgba(0, 0, 0, 0.25),
        0 -12px 30px rgba(0, 0, 0, 0.12),
        0 4px 6px rgba(0, 0, 0, 0.12),
        0 12px 13px rgba(0, 0, 0, 0.17),
        0 -3px 5px rgba(0, 0, 0, 0.09);
}

/* Scrollbar Styling */
::-webkit-scrollbar {
    width: 5px;
}

::-webkit-scrollbar-track {
    background: #f1f1f1;
    border-radius: 10px;
    box-shadow: inset 0 0 5px grey;
}

::-webkit-scrollbar-thumb {
    background: #000;
    border-radius: 10px;
}

.res-sidebar {
    display: none !important;
}
```

```css
/* Responsive Styling */
@media screen and (max-width: 767px) {
    .lgnd1 {
        width: 100% !important;
    }

    .head {
        flex-direction: column !important;
        margin-top: 5% !important;
    }

    .head-info,
    .head-info1,
    .head-info2 {
        width: 90% !important;
    }

    .head-info1 {
        display: flex !important;
        justify-content: space-between !important;
        margin-bottom: 5% !important;
        margin-top: 5% !important;
    }

    .sideblack {
        display: none !important;
    }

    .res-sidebar {
        align-items: center !important;
        display: flex !important;
        justify-content: space-evenly !important;
    }

    .all-courses {
        height: 300px !important;
```

```
        margin-left: 0 !important;
        width: 100% !important;
    }

    .all-courses-1 {
        margin-top: 15% !important;
    }

    .all-course-side,
    .upcoming-res {
        display: none !important;
    }

    .upcoming-res1 {
        width: 70% !important;
    }

    .set-res1 {
        flex-direction: column !important;
    }

    .headres {
        left: 0 !important;
    }

    .noti-head {
        flex-direction: column !important;
    }

    .btn-noti {
        width: 60% !important;
    }

    .all-course-div {
        flex-direction: column !important;
        height: auto !important;
```

```
    }

    .all-course-div1 {
        height: auto !important;
        width: 100% !important;
    }
}
```

3. Environment Configuration (`.env`)

This file holds your environment variables, such as database connection strings and API keys.

```
MONGO_URI=mongodb://localhost:27017/onlinelearningplatformusingmern
PORT=5000
```

4. Example `package.json` for Backend

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node app.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.0.2",
```

```
    "express": "^4.18.1",
    "mongoose": "^6.5.4",
    "otp-generator": "^4.0.0",
    "request": "^2.88.2"
  }
}
```

5. Running the Project

1. Backend Setup:
   - Navigate to the `backend` folder.
   - Install dependencies: `npm install`
   - Start the server in development mode: `npm run dev`

2. Frontend Setup:
   - Navigate to the `frontend` folder.
   - Install dependencies: `npm install`
   - Start the front-end development server: `npm start`

RESULTS

The launch of the interactive online learning platform showcases the prowess of the MERN stack in delivering cutting-edge, scalable learning experiences. Notable achievements include:

1. Immersive User Experience: The platform offers an intuitive and engaging interface, ensuring seamless navigation across devices, and making learning accessible from anywhere.

2. Efficient Content Management: MongoDB's flexibility facilitates the storage and retrieval of diverse learning materials, enabling educators to create and manage courses easily.

3. Scalability: The platform is built to accommodate a growing number of learners and courses without compromising performance, ensuring uninterrupted access to educational resources.

4. Data Security: Robust security measures, such as user authentication and data encryption, safeguard learner information, fostering a secure learning environment.

CONCLUSION

This project showcases the potential of the MERN stack as an exceptional platform for developing online learning platforms. Utilizing MongoDB, Express.js, React.js, and Node.js, the project offers a scalable, efficient, and secure web application. The structured approach ensures the reliability and performance of the platform, from architecture design to testing. Future improvements may include real-time data updates through WebSockets, enabling seamless interactions between learners and educators. Additionally, optimizing the database structure can enhance query performance, leading to faster and more efficient data retrieval for personalized learning experiences.

REFERENCES

1. MongoDB - Official Documentation: https://www.mongodb.com/docs/
2. Express.js - Official Documentation: https://expressjs.com/
3. React.js - Official Documentation: https://react.dev/
React Router Documentation (for handling routing in React): https://reactrouter.com/
4. Node.js - Official Documentation: https://nodejs.org/en/docs/
Getting Started with Node.js: https://nodejs.dev/en/learn/