BOSSCODER ACADEMY

30-DAY

Challenge to Learn

MERN Stack

DAY 1

- Set up a development environment with Node.js, MongoDB, and React
- Create a basic Express server and a React application
- Connect the React frontend to the Express backend

Resources/Tips

- Node.js: https://nodejs.org/
- MongoDB: https://www.mongodb.com/
- Create React App: https://create-react-app.dev/

Daily Checklist

- [x] How would you create a new Express project?
- [ ] How do you start a React application using Create React App?
- [ ] Write a basic Express route that sends a "Hello, World!" response.
- [ ] Build a React component that displays a simple message on the screen.
- [ ] Connect the React frontend to the Express backend and fetch data from an API endpoint.

DAY 2

- Learn the fundamentals of React, including components, state, and props
- Build reusable React components
- Use React's virtual DOM for efficient rendering

Resources/Tips

- React Documentation: https://reactjs.org/docs/getting-started.html
- React Component Lifecycle: https://reactjs.org/docs/react-component.html

Daily Checklist

- [x] Create a React component that accepts a prop and displays it on the screen.
- [ ] Build a reusable React component, such as a button or input field.
- [ ] Write a function in a React component that modifies the component's state.
- [ ] Update a React component to handle user interactions, such as button clicks.
- [ ] Explain the difference between state and props in React and provide an example for each.

DAY 3

- Create a RESTful API using Express
- Define API routes for CRUD operations (Create, Read, Update, Delete)
- Use Postman or Insomnia to test API endpoints

Resources/Tips

- Express Documentation: https://expressjs.com/
- Postman: https://www.postman.com/
- Insomnia: https://insomnia.rest/

Daily Checklist

- [x] Create an Express route that handles a POST request to create a new resource.
- [ ] Write an Express route that retrieves a specific resource using a GET request.
- [ ] Implement an Express route that updates a resource with a PUT request.
- [ ] Build an Express route that deletes a resource using a DELETE request.
- [ ] Test your API endpoints using Postman or Insomnia and verify the expected responses.

DAY 4

- Set up a MongoDB database and connect it to your Express application
- Learn MongoDB CRUD operations (Create, Read, Update, Delete)
- Use Mongoose to interact with the MongoDB database

Resources/Tips

- MongoDB Atlas (Cloud Database): https://www.mongodb.com/cloud/atlas
- Mongoose Documentation: https://mongoosejs.com/

Daily Checklist

- [x] Create a MongoDB database and connect it to your Express application.
- [x] Define a Mongoose schema for a resource in your application.
- [x] Write Mongoose queries to create, read, update, and delete resources.
- [x] Implement an Express route that uses Mongoose to interact with the database.
- [x] Test your database operations by sending API requests and verifying the data in the MongoDB collection.

DAY 5

- Implement data fetching from the backend in React using Axios or Fetch API
- Update React components to display data from API responses
- Handle loading states and error handling in React

Resources/Tips

- Axios: https://axios-http.com/
- Fetch API: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Daily Checklist

- [x] Write an Axios or Fetch API request to fetch data from an API endpoint in your React application.
- [ ] Update a React component to display data fetched from the backend.
- [ ] Implement a loading state in a React component while waiting for the API response.
- [ ] Handle error cases in a React component when the API request fails.
- [ ] Build a form in React that submits data to an API endpoint and handles the response.

DAY 6

- Learn about React Router for client-side routing
- Set up routes in a React application using React Router
- Implement navigation between different pages in React

Resources/Tips

- React Router: https://reactrouter.com/

Daily Checklist

- [x] Set up React Router in your application to define routes for different pages.
- [ ] Create a navigation menu with links to different pages using React Router.
- [ ] Build multiple React components, each associated with a different route.
- [ ] Implement a nested route structure in your React application.
- [ ] Add a 404 page for handling invalid routes in your React Router configuration.

DAY 7

- Implement form handling and validation in React
- Use form libraries (e.g., Formik, React Hook Form) for easier form management
- Validate form inputs and display error messages

Resources/Tips

- Formik: https://formik.org/
- React Hook Form: https://react-hook-form.com/

Daily Checklist

- [x] Build a form in React using Formik or React Hook Form.
- [ ] Validate form inputs for required fields, email format, or custom validation rules.
- [ ] Display error messages for invalid form inputs.
- [ ] Handle form submission and send data to the backend API.
- [ ] Reset the form after successful submission or cancellation.

DAY 8

- Implement pagination and sorting in React
- Retrieve paginated data from the backend API
- Allow users to sort data based on different criteria

Resources/Tips

- React Paginate: https://www.npmjs.com/package/react-paginate
- Backend API Pagination Example: https://docs.mongodb.com/manual/tutorial/pagination-example-with-compound-sort/

Daily Checklist

- [x] Fetch paginated data from the backend API and display it in a React component.
- [ ] Implement pagination controls in your React application to navigate between pages.
- [ ] Add sorting functionality to the React component to allow users to sort data based on different criteria.
- [ ] Update the API endpoints to handle pagination and sorting parameters.
- [ ] Test the pagination and sorting functionality by fetching data from the backend and verifying the results.

DAY 9

- Implement user authentication and registration in the MERN stack
- Set up user authentication using JWT (JSON Web Tokens)
- Enable user registration and login functionality

Resources/Tips

- JWT (JSON Web Tokens): https://jwt.io/

Daily Checklist

- [x] Implement user registration functionality by creating an API endpoint that stores user information in the database.
- [ ] Set up user login functionality by verifying credentials and generating a JWT token.
- [ ] Protect specific routes in your backend API by checking the validity of the JWT token.
- [ ] Update the React frontend to handle user registration and login forms.
- [ ] Test the user authentication and registration flows, ensuring successful login and access to protected routes.

DAY 10

- Implement user profiles and account management
- Allow users to view and edit their profile information
- Implement password reset and change password functionality

Resources/Tips

- MongoDB Update Operators: https://docs.mongodb.com/manual/reference/operator/update/
- Express middleware for authentication: https://expressjs.com/en/guide/using-middleware.html

Daily Checklist

- [x] Create an API endpoint that allows users to view their profile information.
- [ ] Implement an API endpoint that allows users to update their profile information.
- [ ] Build a React component that displays the user's profile information and allows them to edit it.
- [ ] Implement password reset functionality by sending a password reset link to the user's email.
- [ ] Add a "Change Password" feature that allows users to update their password with a new one.

DAY 11

- Implement file uploads in the MERN stack
- Allow users to upload files (e.g., images) to the server
- Store the uploaded files and serve them when requested

Resources/Tips

- multer (Node.js middleware for handling file uploads): https://www.npmjs.com/package/multer

Daily Checklist

- [x] Set up multer to handle file uploads in your Express application.
- [ ] Create an API endpoint that accepts file uploads and saves them to a designated directory.
- [ ] Modify your React frontend to include a file upload form and send files to the backend API.
- [ ] Implement an API endpoint that serves the uploaded files when requested.
- [ ] Test the file upload functionality by uploading files from the frontend and verifying their availability on the server.

DAY 12

- Implement real-time features with WebSocket communication
- Set up WebSocket server using libraries like Socket.IO
- Update React components to handle real-time updates

Resources/Tips

- Socket.IO: https://socket.io/

Daily Checklist

- [x] Set up a WebSocket server using Socket.IO in your Express application.
- [ ] Update a React component to establish a WebSocket connection and receive real-time updates from the server.
- [ ] Emit WebSocket events from the server and handle them in the React component.
- [ ] Implement a chat feature using WebSocket communication in your MERN stack application.
- [ ] Test the real-time functionality by simulating multiple users and verifying that updates are received in real-time.

DAY 13

- Implement data validation and sanitization to improve security
- Use libraries like Yup or Validator.js for validation
- Sanitize user input to prevent vulnerabilities like cross-site scripting (XSS)

Resources/Tips

- Yup: https://www.npmjs.com/package/yup
- Validator.js: https://www.npmjs.com/package/validator

Daily Checklist

- [x] Use Yup or Validator.js to validate form inputs in your React application.
- [ ] Implement server-side validation using Yup or a similar library to validate API request payloads.
- [ ] Sanitize user input to prevent cross-site scripting (XSS) vulnerabilities.
- [ ] Handle validation errors in your React components and display appropriate error messages.
- [ ] Test the validation and sanitization functionality by submitting invalid data and ensuring it is rejected or sanitized correctly.

DAY 14

- Implement role-based access control (RBAC) for different user roles
- Restrict access to certain routes or features based on user roles
- Manage user roles and permissions in the backend

Resources/Tips

- Access Control Lists (ACL): https://en.wikipedia.org/wiki/Access-control_list

Daily Checklist

- [x] Create different user roles in your MERN stack application (e.g., admin, user, guest).
- [x] Implement server-side logic to enforce RBAC by checking user roles and permissions.
- [x] Restrict access to certain routes or features based on user roles.
- [x] Build a React component that displays different content or features based on the user's role.
- [x] Test the role-based access control by logging in with different user accounts and verifying access to restricted routes or features.

DAY 15

- Implement internationalization (i18n) for multi-language support
- Set up language files and translations
- Update React components to display content in different languages

Resources/Tips

- react-i18next: https://react.i18next.com/

Daily Checklist

- [x] Set up language files (e.g., JSON or YAML) for different languages in your React application.
- [ ] Implement a language switcher component that allows users to switch between languages.
- [ ] Update React components to display content based on the selected language using i18n libraries like react-i18next.
- [ ] Create translation files for commonly used phrases or labels in your application.
- [ ] Test the multi-language support by switching between different languages and ensuring the content is displayed correctly.

DAY 16

- Implement data caching for improved performance
- Use caching mechanisms like Redis or Memcached
- Cache frequently accessed data in your MERN stack application

Resources/Tips

- Redis: https://redis.io/
- Memcached: https://memcached.org/

Daily Checklist

- [x] Set up Redis or Memcached as a caching mechanism in your backend application.
- [ ] Identify frequently accessed data in your MERN stack application that can be cached.
- [ ] Implement caching logic in your backend API to cache data and serve it from the cache when available.
- [ ] Test the caching functionality by monitoring the network requests and verifying the data is served from the cache when appropriate.
- [ ] Implement cache invalidation strategies to ensure data consistency and freshness.

DAY 17

- Implement server-side rendering (SSR) for improved performance and SEO
- Use libraries like Next.js or React Server Components
- Convert React components to server-rendered components

Resources/Tips

- Next.js: https://nextjs.org/
- React Server Components: https://reactjs.org/blog/2020/12/21/data-fetching-with-react-server-components.html

Daily Checklist

- [x] Set up a Next.js application or a similar SSR framework for your React frontend.
- [ ] Convert React components to server-rendered components using Next.js or React Server Components.
- [ ] Test the server-side rendering by checking the source code of the rendered page and ensuring the initial load time is improved.
- [ ] Implement data fetching in server-rendered components to pre-render data before sending it to the client.
- [ ] Verify the improved performance and SEO benefits by comparing the server-rendered version of your application with the client-side rendered version.

DAY 18

- Implement unit testing for your MERN stack application
- Write test cases using libraries like Jest or Mocha
- Test both frontend React components and backend API endpoints

Resources/Tips

- Jest: https://jestjs.io/
- Mocha: https://mochajs.org/

Daily Checklist

- [x] Set up a testing framework (e.g., Jest or Mocha) for your MERN stack application.
- [ ] Write unit tests for React components, covering different use cases and edge cases.
- [ ] Test API endpoints using libraries like Supertest to ensure they return the expected responses.
- [ ] Mock external dependencies and APIs in your tests to isolate your code.
- [ ] Run the test suite and ensure all tests pass, providing confidence in the stability and correctness of your application.

DAY 19

- Implement end-to-end testing for your MERN stack application
- Use tools like Cypress or Puppeteer for UI testing
- Write test scenarios to cover user interactions and workflows

Resources/Tips

- Cypress: https://www.cypress.io/
- Puppeteer: https://pptr.dev/

Daily Checklist

- [x] Set up an end-to-end testing framework (e.g., Cypress or Puppeteer) for your MERN stack application.
- [ ] Write test scenarios to cover common user interactions and workflows in your application.
- [ ] Simulate user actions (e.g., clicking buttons, filling forms) and verify the expected outcomes.
- [ ] Test error handling and edge cases to ensure your application behaves correctly in all scenarios.
- [ ] Run the end-to-end test suite and ensure all tests pass, providing confidence in the functionality and usability of your application.

DAY 20

- Optimize the performance of your MERN stack application
- Identify performance bottlenecks and apply optimizations
- Improve loading times and overall user experience

Resources/Tips

- Performance Optimization Guide: https://developers.google.com/web/fundamentals/performance

Daily Checklist

- [x] Analyze your MERN stack application's performance using tools like Lighthouse or Chrome DevTools.
- [ ] Identify performance bottlenecks, such as slow database queries or inefficient React component rendering.
- [ ] Apply optimizations like database indexing, query optimization, or React component memoization to improve performance.
- [ ] Measure and compare the loading times before and after optimizations to assess the improvement.
- [ ] Test the user experience and ensure the application feels responsive and performs well under different usage scenarios.

DAY 21

- Implement error logging and monitoring in your MERN stack application
- Use tools like Sentry or LogRocket for error tracking
- Monitor and analyze application errors for debugging and improving user experience

Resources/Tips

- Sentry: https://sentry.io/
- LogRocket: https://logrocket.com/

Daily Checklist

- [x] Set up error logging and monitoring using Sentry or LogRocket in your application.
- [ ] Capture and log errors from both the frontend and backend to identify and debug issues.
- [ ] Monitor error trends and analyze error logs to prioritize bug fixes and improvements.
- [ ] Use error monitoring tools to gain insights into user experience and identify areas for optimization.
- [ ] Implement error reporting and feedback mechanisms for users to report issues encountered.

DAY 22

- Implement performance monitoring and optimization for your MERN stack application
- Use tools like New Relic or Datadog to monitor application performance
- Identify performance bottlenecks and optimize critical areas

Resources/Tips

- New Relic: https://newrelic.com/
- Datadog: https://www.datadoghq.com/

Daily Checklist

- [x] Set up performance monitoring using tools like New Relic or Datadog in your application.
- [ ] Monitor critical performance metrics like response time, CPU usage, memory consumption, and database queries.
- [ ] Identify performance bottlenecks using monitoring data and apply optimizations to improve critical areas.
- [ ] Continuously monitor and analyze performance to ensure your application meets performance requirements.
- [ ] Optimize resource-intensive operations and database queries for better scalability and responsiveness.

DAY 23

- Implement automated testing and continuous integration (CI) for your MERN stack application
- Use tools like Jenkins, Travis CI, or GitHub Actions for CI/CD
- Set up a pipeline to run tests and deploy the application automatically

Resources/Tips

- Jenkins: https://www.jenkins.io/
- Travis CI: https://travis-ci.com/
- GitHub Actions: https://github.com/features/actions

Daily Checklist

- [x] Set up a CI/CD pipeline using tools like Jenkins, Travis CI, or GitHub Actions.
- [ ] Configure the pipeline to run automated tests on each code commit or pull request.
- [ ] Ensure the pipeline deploys the application automatically to a staging or production environment upon successful tests.
- [ ] Monitor and analyze the pipeline's output and ensure it provides actionable feedback for failed tests or deployment issues.
- [ ] Continuously improve the pipeline by adding additional checks, code analysis, or performance benchmarks.

DAY 24

- Implement data backup and recovery mechanisms for your MERN stack application
- Set up regular backups of your database and application data
- Define a recovery plan to restore data in case of failures or data loss

Resources/Tips

- MongoDB Backup and Restore: https://docs.mongodb.com/manual/core/backups/
- AWS S3 for file backups: https://aws.amazon.com/s3/

Daily Checklist

- [x] Set up regular backups for your database using tools like mongodump or automated backup solutions.
- [ ] Implement backup mechanisms for application files (e.g., uploaded images) using cloud storage services like AWS S3.
- [ ] Test the data recovery process by restoring backups to a different environment or verifying file retrieval from the cloud storage.
- [ ] Document the data backup and recovery procedures for future reference or in case of emergencies.
- [ ] Regularly validate and update the backup and recovery plan to ensure it remains effective.

DAY 25

- Implement performance optimization techniques for your MERN stack application
- Apply techniques like code splitting, lazy loading, and asset optimization
- Optimize frontend and backend code for better performance

Resources/Tips

- Code Splitting in React: https://reactjs.org/docs/code-splitting.html
- Webpack for asset optimization: https://webpack.js.org/

Daily Checklist

- [x] Apply code splitting techniques in your React application to split bundled code into smaller chunks.
- [ ] Implement lazy loading for components or routes that are not immediately required for the initial page load.
- [ ] Optimize frontend assets (e.g., images, stylesheets) by compressing, minifying, or using modern formats like WebP.
- [ ] Optimize database queries and backend code for better performance and reduced response times.
- [ ] Measure and compare performance metrics before and after optimizations to assess the improvements.

DAY 26

- Implement user authentication and authorization in your MERN stack application
- Use libraries like Passport.js or JSON Web Tokens (JWT) for authentication
- Define user roles and permissions for authorization

Resources/Tips

- Passport.js: http://www.passportjs.org/
- JWT: https://jwt.io/

Daily Checklist

- [x] Implement user registration and login functionality using Passport.js or JWT.
- [ ] Set up authentication middleware to protect routes that require user authentication.
- [ ] Define user roles and permissions to control access to specific features or resources.
- [ ] Implement authorization checks in your API endpoints or React components to ensure only authorized users can perform certain actions.
- [ ] Test the authentication and authorization flows by simulating different user scenarios and verifying access control.

DAY 27

- Implement email integration and notifications in your MERN stack application
- Use libraries like Nodemailer or SendGrid for sending emails
- Send transactional emails or notifications to users

Resources/Tips

- Nodemailer: https://nodemailer.com/
- SendGrid: https://sendgrid.com/

Daily Checklist

- [x] Set up email integration using Nodemailer or a similar library in your backend application.
- [ ] Create email templates and send transactional emails (e.g., account verification, password reset) to users.
- [ ] Implement email notifications for events or updates in your application.
- [ ] Test the email integration by sending emails to different email addresses and verifying their delivery.
- [ ] Monitor and log email sending status or failures for debugging and tracking purposes.

DAY 28

- Implement search functionality in your MERN stack application
- Use technologies like Elasticsearch or MongoDB's text search for search capabilities
- Enable users to search and filter data based on specific criteria

Resources/Tips

- Elasticsearch: https://www.elastic.co/elasticsearch/
- MongoDB Text Search: https://docs.mongodb.com/manual/text-search/

Daily Checklist

- [x] Set up a search index using Elasticsearch or enable text search in MongoDB for the relevant collections.
- [ ] Implement search functionality in your frontend, allowing users to enter search queries and retrieve matching results.
- [ ] Apply filters and additional search criteria to refine search results based on user requirements.
- [ ] Optimize search queries and indexing for better search performance.
- [ ] Test the search functionality with different search queries and verify the accuracy and relevancy of the results.

DAY 29

- Implement user feedback and rating features in your MERN stack application
- Allow users to provide feedback or ratings for various entities (e.g., products, services)
- Display and aggregate user feedback to enhance user experience and decision-making

Resources/Tips

- React Rating Libraries: https://blog.bitsrc.io/11-react-rating-libraries-for-building-star-rating-component-in-reactjs-670f38a7e160

Daily Checklist

- [ ] Add a feedback or rating component to your React application to collect user feedback or ratings.
- [ ] Implement backend API endpoints to store and retrieve user feedback or ratings for specific entities.
- [ ] Display average ratings and user feedback on relevant pages or components.
- [ ] Provide the ability for users to update or delete their feedback/ratings if applicable.
- [ ] Test the feedback and rating features by submitting feedback and verifying its storage and display.

DAY 30

- Finalize your MERN stack application and deploy it to a production environment
- Perform final testing and bug fixing
- Optimize the deployment for performance, security, and scalability

Daily Checklist

- [x] Conduct a thorough testing phase to identify and fix any remaining bugs or issues.
- [ ] Optimize the application's performance, security, and scalability in preparation for deployment.
- [ ] Set up a production environment (e.g., AWS, Heroku, or your preferred hosting provider) and deploy your application.
- [ ] Monitor and ensure the stability and availability of your deployed application.
- [ ] Celebrate your achievement and share your MERN stack application with others!

Did you find it Useful?

Leave a comment!

Alamin CodePapa

@CodePapa360

FOLLOW FOR MORE

Like Comment Repost