notes

July 27, 2025

1 FullStack Backend Project - Node.js & Express

1.1 Description

This is a backend project for practicing full-stack development with Node.js, Express, and MongoDB. The project focuses on building an authentication system with user verification and password reset functionality.

1.2 Technologies Used

- **Node.js** - JavaScript runtime
- **Express.js** - Web framework
- **MongoDB** - Database
- **Mongoose** - MongoDB object modeling
- **dotenv** - Environment variable management
- **nodemon** - Development tool for auto-reloading

1.3 Project Structure

```
FULLSTACK2/
  index.js          # Main server file
  package.json       # Dependencies and scripts
  utils/
     db.js         # Database connection
  .env              # Environment variables
```

1.4 Getting Started

1.4.1 Installation

```
# Install dependencies
npm install

# Start development server
npm run dev
```

1.4.2 Environment Variables

Create a `.env` file in the root directory:

```
PORT=3000
MONGO_URL=mongodb+srv://admin:<db_password>@cluster0.f7wudhx.mongodb.net/
BASE_URL=http://localhost:3000
```

1.5 Available Ports

The following ports are commonly used and available: - 3002 - 4000 - 5000 - 5173 - 8080 - 8000

1.6 HTTP Request Types

The server handles four main types of HTTP requests: - **GET** - Retrieve data - **POST** - Create new data - **PUT** - Update existing data - **DELETE** - Remove data

1.7 Server Configuration

1.7.1 Express Middleware

```
// Parse JSON data
app.use(express.json())

// Parse URL-encoded data
app.use(express.urlencoded({extended: true}))

// CORS configuration for frontend-backend communication
// Restricts direct client access to backend
```

1.7.2 Port Configuration

```
const port = process.env.PORT || 4000;
```

Note: In production, the server allocates the port automatically. Using `process.env.PORT` is a best practice for deployment on platforms like Render, Digital Ocean, or AWS.

1.8 Database Connection

1.8.1 MongoDB with Mongoose

- **Direct MongoDB connection**: Possible but not recommended
- **Mongoose**: Acts as a middleman between backend and MongoDB
- **Benefits**: Easier to use than direct MongoDB queries

1.8.2 Connection Requirements

1. Database setup
2. IP whitelisting
3. Username and password
4. **Important**: Avoid special characters in MongoDB user passwords to prevent connection errors

1.8.3 Mongoose Features

- Automatic timestamp fields when `timestamp: true` is set:
  - `createdAt`
  - `updatedAt`

1.9 Authentication System

1.9.1 User Schema Fields

```
{
  name: String,
  email: String,
  password: String,
  role: String, // e.g., "teacher", "student" for LMS
  isVerified: Boolean,
  verificationToken: String,
  passwordResetToken: String,
  passwordResetExpires: Date,
  createdAt: Date,
  updatedAt: Date
}
```

1.9.2 Password Reset Flow

1. **Step 1**: User sends password reset request to `/reset` route with email
2. **Step 2**: Server checks if email exists in database
   - If not found: Send "account doesn't exist" message
   - If found: Proceed to step 3
3. **Step 3**: Generate random token (e.g., "jnegyqhudseq")
4. **Step 4**: Send token copy to user and store copy in database
5. **Step 5**: User visits `/reset-password` route with token
6. **Step 6**: Server validates token against database
7. **Step 7**: If valid, allow password reset

Token Security: Tokens expire after 5-10 minutes using `passwordResetExpires` field.

1.9.3 User Verification Flow

1. **Step 1**: User signs up and receives verification email/OTP
2. **Step 2**: User visits `/verify` page and enters OTP
3. **Step 3**: Server compares OTP with stored `verificationToken`
4. **Step 4**: If valid, mark user as verified

1.10 Security Considerations

1.10.1 Environment Variables

- Store sensitive data in `.env` file
- Never commit `.env` to version control
- Use `.env.sample` for sharing environment structure

1.10.2 CORS (Cross-Origin Resource Sharing)

- **Problem**: Separation of frontend and backend (unlike traditional PHP/Laravel monoliths)
- **Solution**: Implement CORS to restrict direct client access
- **Goal**: Only allow frontend to communicate with backend

1.11 Development Tools

1.11.1 Dependencies Types

1. **Crucial Dependencies**: Required for production (express, mongoose, dotenv)
2. **Development Dependencies**: Only needed during development (nodemon)

1.11.2 Nodemon

- Monitors Node.js applications
- Automatically reloads server on file changes
- Essential for development workflow

1.12 Routes and Controllers

1.12.1 Route Structure

```
//  Incorrect - missing leading slash
app.get("kusum", (req, res) => {
    res.send("kusum!")
});

//  Correct - with leading slash
app.get("/kusum", (req, res) => {
    res.send("kusum maa!")
});
```

1.12.2 Controllers

Controllers handle the response logic:

```
// This function is the controller
(req, res) => {
    res.send("kusum maa!")
}
```

Best Practice: Separate routes and controllers for better code organization.

1.13 Deployment Notes

- Use environment variables for configuration
- Server automatically allocates ports in production
- Ensure all sensitive data is in environment variables
- Test CORS configuration with frontend

1.14 Learning Objectives

- Understand backend-frontend separation
- Implement secure authentication flows
- Work with MongoDB and Mongoose
- Handle environment variables securely
- Implement token-based verification systems