Full-Stack JavaScript Mastery: MERN to Millions

Complete MERN Stack Guide with 3 Production Projects

> "The best way to learn to code is to code." - Quincy Larson

---

Table of Contents

Chapter 1: MERN Stack Foundation

Why MERN Stack Dominates the Market

Industry Statistics: - 68% of developers use JavaScript for full-stack development - MERN developers average $85K-180K salary range - 95% of Fortune 500 companies use Node.js in production - React has 74% market share among frontend frameworks - MongoDB powers 50% of modern web applications

MERN Stack Components: - **M**ongoDB: NoSQL database for flexible data storage - **E**xpress.js: Fast, minimalist web framework for Node.js - **R**eact: Component-based frontend library - **N**ode.js: JavaScript runtime for server-side development

Complete Development Environment Setup

Prerequisites Installation:

```
# Node.js (LTS version recommended)
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
nvm install --lts
nvm use --lts

# MongoDB Community Edition
brew tap mongodb/brew
brew install mongodb-community@7.0
brew services start mongodb/brew/mongodb-community

# Essential development tools
npm install -g nodemon create-react-app npm-check-updates
npm install -g @mongodb-js/compass # MongoDB GUI
```

Project Structure Template:

```
mern-project/
├── client/                 # React frontend application
│   ├── public/
│   ├── src/
│   │   ├── components/     # Reusable UI components
│   │   ├── pages/         # Page-level components
│   │   ├── hooks/         # Custom React hooks
│   │   ├── context/       # Global state management
│   │   ├── services/      # API calls and external services
│   │   ├── utils/         # Utility functions
│   │   └── styles/        # CSS and styling files
│   ├── package.json
│   └── .env
├── server/                 # Express.js backend API
│   ├── controllers/        # Request handling logic
│   ├── models/            # MongoDB schema definitions
│   ├── routes/            # API route definitions
│   ├── middleware/        # Express middleware functions
│   ├── config/            # Database and app configuration
│   ├── utils/             # Server-side utilities
│   ├── package.json
│   └── .env
├── shared/                 # Shared utilities and types
├── docs/                  # Project documentation
└── README.md
```

Essential Dependencies:

```
 // server/package.json
{
  "dependencies": {
    "express": "^4.18.2",
    "mongoose": "^7.5.0",
    "cors": "^2.8.5",
    "helmet": "^7.0.0",
    "morgan": "^1.10.0",
    "dotenv": "^16.3.1",
    "bcryptjs": "^2.4.3",
    "jsonwebtoken": "^9.0.2",
    "express-validator": "^7.0.1",
    "multer": "^1.4.5-lts.1",
    "compression": "^1.7.4",
    "express-rate-limit": "^6.10.0"
  }
}

// client/package.json
{
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.15.0",
    "axios": "^1.5.0",
    "@tanstack/react-query": "^4.32.6",
    "react-hook-form": "^7.45.4",
    "@hookform/resolvers": "^3.3.1",
    "yup": "^1.3.2",
    "styled-components": "^6.0.7",
    "react-toastify": "^9.1.3",
    "date-fns": "^2.30.0"
  }
}
```

Chapter 2: Project #1 - Task Management SaaS

Project Overview and Business Model

Application Features: - User authentication and authorization - Project and task management - Team collaboration and permissions - Real-time updates and notifications - File uploads and attachments - Analytics and reporting dashboard - Subscription-based monetization

Technology Implementation: - Frontend: React with Context API for state management - Backend: Express.js with JWT authentication - Database: MongoDB with Mongoose ODM - Real-time: Socket.io for live updates - File Storage: AWS S3 or local file system - Payment: Stripe for subscription management

Backend Implementation

Database Schema Design:

```
 // models/User.js
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: [true, 'Username is required'],
    unique: true,
    trim: true,
    minlength: 3,
    maxlength: 30
  },
  email: {
    type: String,
    required: [true, 'Email is required'],
    unique: true,
    lowercase: true,
    match: [/^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/, 'Invalid email']
  },
  password: {
    type: String,
    required: [true, 'Password is required'],
    minlength: 6
  },
  avatar: {
    type: String,
    default: null
  },
  role: {
    type: String,
    enum: ['user', 'admin', 'manager'],
    default: 'user'
  },
  subscription: {
    plan: {
      type: String,
      enum: ['free', 'pro', 'enterprise'],
      default: 'free'
    },
    status: {
      type: String,
      enum: ['active', 'inactive', 'cancelled'],
      default: 'active'
    },
    stripeCustomerId: String,
    stripeSubscriptionId: String,
    currentPeriodEnd: Date
  },
  preferences: {
    theme: {
      type: String,
      enum: ['light', 'dark'],
      default: 'light'
    },
    emailNotifications: {
      type: Boolean,
      default: true
    },
    timezone: {
      type: String,
      default: 'UTC'
    }
  },
  lastLogin: Date,
  isActive: {
    type: Boolean,
    default: true
  }
}, {
  timestamps: true
});
```

```
// Hash password before saving
userSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();

  try {
    const salt = await bcrypt.genSalt(12);
    this.password = await bcrypt.hash(this.password, salt);
    next();
  } catch (error) {
    next(error);
  }
});

// Compare password method
userSchema.methods.comparePassword = async function(candidatePassword) {
  return bcrypt.compare(candidatePassword, this.password);
};

// Remove sensitive data when converting to JSON
userSchema.methods.toJSON = function() {
  const user = this.toObject();
  delete user.password;
  delete user.subscription.stripeCustomerId;
  delete user.subscription.stripeSubscriptionId;
  return user;
};

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

// models/Project.js
const projectSchema = new mongoose.Schema({
  title: {
    type: String,
    required: [true, 'Project title is required'],
    trim: true,
    maxlength: 100
  },
  description: {
    type: String,
    maxlength: 500
  },
  owner: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  team: [{
    user: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    },
    role: {
      type: String,
      enum: ['viewer', 'editor', 'admin'],
      default: 'editor'
    },
    joinedAt: {
      type: Date,
      default: Date.now
    }
  }],
  status: {
    type: String,
    enum: ['planning', 'active', 'on-hold', 'completed', 'archived'],
    default: 'planning'
  },
  priority: {
    type: String,
    enum: ['low', 'medium', 'high', 'urgent'],
    default: 'medium'
  },
  startDate: Date,
  dueDate: Date,
  completedAt: Date,
  tags: [String],
  color: {
    type: String,
    default: '#3498db'
  },
  isArchived: {
    type: Boolean,
    default: false
  }
}, {
  timestamps: true
});

// models/Task.js
const taskSchema = new mongoose.Schema({
  title: {
    type: String,
    required: [true, 'Task title is required'],
    trim: true,
    maxlength: 200
  },
  description: {
    type: String,
    maxlength: 1000
  },
  project: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Project',
    required: true
  },
  assignee: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  },
  reporter: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  status: {
    type: String,
    enum: ['todo', 'in-progress', 'review', 'done'],
    default: 'todo'
  },
  priority: {
    type: String,
    enum: ['low', 'medium', 'high', 'urgent'],
    default: 'medium'
  },
  labels: [String],
  dueDate: Date,
  estimatedHours: Number,
  actualHours: Number,
  attachments: [{
    filename: String,
    originalName: String,
    mimetype: String,
    size: Number,
    uploadedBy: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User'
    },
    uploadedAt: {
      type: Date,
      default: Date.now
    }
  }],
  comments: [{
    author: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'User',
      required: true
    },
    content: {
      type: String,
      required: true,
      maxlength: 500
    },
    createdAt: {
      type: Date,
      default: Date.now
    }
  }],
  completedAt: Date
}, {
  timestamps: true
});
```

Authentication and Authorization System:

```
 // middleware/auth.js
const jwt = require('jsonwebtoken');
const User = require('../models/User');

const authMiddleware = async (req, res, next) => {
  try {
    const token = req.header('Authorization')?.replace('Bearer ', '');

    if (!token) {
      return res.status(401).json({
        error: 'Access denied. No token provided.'
      });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = await User.findById(decoded.id).select('-password');

    if (!user || !user.isActive) {
      return res.status(401).json({
        error: 'Invalid token or user deactivated.'
      });
    }

    req.user = user;
    next();
  } catch (error) {
    res.status(401).json({ error: 'Invalid token.' });
  }
};

const roleMiddleware = (...roles) => {
  return (req, res, next) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({
        error: 'Access denied. Insufficient permissions.'
      });
    }
    next();
  };
};

module.exports = { authMiddleware, roleMiddleware };

// controllers/authController.js
const User = require('../models/User');
const jwt = require('jsonwebtoken');
const { validationResult } = require('express-validator');

const generateToken = (userId) => {
  return jwt.sign({ id: userId }, process.env.JWT_SECRET, {
    expiresIn: process.env.JWT_EXPIRES_IN || '7d'
  });
};

const register = async (req, res) => {
  try {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({
        error: 'Validation failed',
        details: errors.array()
      });
    }

    const { username, email, password } = req.body;

    // Check if user already exists
    const existingUser = await User.findOne({
      $or: [{ email }, { username }]
    });

    if (existingUser) {
      return res.status(400).json({
        error: 'User already exists with this email or username'
      });
    }

    // Create new user
    const user = new User({ username, email, password });
    await user.save();

    // Generate token
    const token = generateToken(user._id);

    res.status(201).json({
      message: 'User registered successfully',
      token,
      user: user.toJSON()
    });

  } catch (error) {
    console.error('Registration error:', error);
    res.status(500).json({ error: 'Server error during registration' });
  }
};

const login = async (req, res) => {
  try {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({
        error: 'Validation failed',
        details: errors.array()
      });
    }
    const { email, password } = req.body;

    // Find user and include password for comparison
    const user = await User.findOne({ email }).select('+password');

    if (!user || !user.isActive) {
      return res.status(401).json({
        error: 'Invalid credentials or account deactivated'
      });
    }

    // Check password
    const isPasswordValid = await user.comparePassword(password);
    if (!isPasswordValid) {
      return res.status(401).json({
        error: 'Invalid credentials'
      });
    }

    // Update last login
    user.lastLogin = new Date();
    await user.save();

    // Generate token
    const token = generateToken(user._id);

    res.json({
      message: 'Login successful',
      token,
      user: user.toJSON()
    });

  } catch (error) {
    console.error('Login error:', error);
    res.status(500).json({ error: 'Server error during login' });
  }
};

module.exports = { register, login };
```

⚛ React Frontend Implementation

Context-Based State Management:

```
// context/AuthContext.js
import React, { createContext, useContext, useReducer, useEffect } from 'react';
import axios from 'axios';

const AuthContext = createContext();
const authReducer = (state, action) => {
  switch (action.type) {
    case 'LOGIN_START':
      return { ...state, loading: true, error: null };
    case 'LOGIN_SUCCESS':
      return {
        ...state,
        loading: false,
        user: action.payload.user,
        token: action.payload.token,
        isAuthenticated: true,
        error: null
      };
    case 'LOGIN_FAILURE':
      return {
        ...state,
        loading: false,
        user: null,
        token: null,
        isAuthenticated: false,
        error: action.payload
      };
    case 'LOGOUT':
      return {
        ...state,
        user: null,
        token: null,
        isAuthenticated: false,
        loading: false,
        error: null
      };
    case 'UPDATE_USER':
      return { ...state, user: { ...state.user, ...action.payload } };
    default:
      return state;
  }
};

const initialState = {
  user: null,
  token: localStorage.getItem('token'),
  isAuthenticated: false,
  loading: false,
  error: null
};

export const AuthProvider = ({ children }) => {
  const [state, dispatch] = useReducer(authReducer, initialState);

  // Set up axios interceptor for automatic token attachment
  useEffect(() => {
    if (state.token) {
      axios.defaults.headers.common['Authorization'] = `Bearer ${state.token}`;
    } else {
      delete axios.defaults.headers.common['Authorization'];
    }
  }, [state.token]);

  // Check token validity on app load
  useEffect(() => {
    const token = localStorage.getItem('token');
    if (token) {
      validateToken(token);
    }
  }, []);

  const validateToken = async (token) => {
    try {
      dispatch({ type: 'LOGIN_START' });
      const response = await axios.get('/api/auth/me', {
        headers: { Authorization: `Bearer ${token}` }
      });

      dispatch({
        type: 'LOGIN_SUCCESS',
        payload: { user: response.data.user, token }
      });
    } catch (error) {
      localStorage.removeItem('token');
      dispatch({ type: 'LOGIN_FAILURE', payload: 'Invalid token' });
    }
  };

  const login = async (email, password) => {
    try {
      dispatch({ type: 'LOGIN_START' });

      const response = await axios.post('/api/auth/login', {
        email,
        password
      });

      const { user, token } = response.data;

      localStorage.setItem('token', token);
      dispatch({ type: 'LOGIN_SUCCESS', payload: { user, token } });

      return { success: true };
    } catch (error) {
      const errorMessage = error.response?.data?.error || 'Login failed';
      dispatch({ type: 'LOGIN_FAILURE', payload: errorMessage });
      return { success: false, error: errorMessage };
    }
  };

  const register = async (userData) => {
    try {
      dispatch({ type: 'LOGIN_START' });

      const response = await axios.post('/api/auth/register', userData);
      const { user, token } = response.data;

      localStorage.setItem('token', token);
      dispatch({ type: 'LOGIN_SUCCESS', payload: { user, token } });

      return { success: true };
    } catch (error) {
      const errorMessage = error.response?.data?.error || 'Registration failed';
      dispatch({ type: 'LOGIN_FAILURE', payload: errorMessage });
      return { success: false, error: errorMessage };
    }
  };

  const logout = () => {
    localStorage.removeItem('token');
    dispatch({ type: 'LOGOUT' });
  };

  const updateUser = (userData) => {
    dispatch({ type: 'UPDATE_USER', payload: userData });
  };

  return (
    <AuthContext.Provider
      value={{
        ...state,
        login,
        register,
        logout,
        updateUser
      }}
    >
      {children}
    </AuthContext.Provider>
  );
};

export const useAuth = () => {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error('useAuth must be used within an AuthProvider');
  }
  return context;
};

// Custom hook for protected routes
export const useRequireAuth = () => {
  const { isAuthenticated, loading } = useAuth();

  useEffect(() => {
    if (!loading && !isAuthenticated) {
      window.location.href = '/login';
    }
  }, [isAuthenticated, loading]);

  return { isAuthenticated, loading };
};
```

Project Management Components:

```
 // components/ProjectDashboard.js
import React, { useState, useEffect } from 'react';
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';
import axios from 'axios';
import styled from 'styled-components';
import { toast } from 'react-toastify';
import ProjectCard from './ProjectCard';
import CreateProjectModal from './CreateProjectModal';
import { useAuth } from '../context/AuthContext';

const DashboardContainer = styled.div`
  padding: 2rem;
  max-width: 1200px;
  margin: 0 auto;
`;

const Header = styled.div`
  display: flex;
  justify-content: between;
  align-items: center;
  margin-bottom: 2rem;
`;

const Title = styled.h1`
  color: #2c3e50;
  font-size: 2.5rem;
  font-weight: 700;
`;

const CreateButton = styled.button`
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  color: white;
  border: none;
  padding: 0.75rem 1.5rem;
  border-radius: 8px;
  font-weight: 600;
  cursor: pointer;
  transition: transform 0.2s ease;

  &:hover {
    transform: translateY(-2px);
  }
`;
```

```
const ProjectGrid = styled.div`
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
  gap: 1.5rem;
  margin-top: 2rem;
`;

const FilterContainer = styled.div`
  display: flex;
  gap: 1rem;
  margin-bottom: 1.5rem;
`;

const FilterButton = styled.button`
  padding: 0.5rem 1rem;
  border: 2px solid #e0e6ed;
  background: ${props => props.active ? '#667eea' : 'white'};
  color: ${props => props.active ? 'white' : '#666'};
  border-radius: 6px;
  cursor: pointer;
  transition: all 0.2s ease;

  &:hover {
    border-color: #667eea;
  }
`;

const ProjectDashboard = () => {
  const { user } = useAuth();
  const [showCreateModal, setShowCreateModal] = useState(false);
  const [filter, setFilter] = useState('all');
  const queryClient = useQueryClient();

  // Fetch projects
  const { data: projects = [], isLoading, error } = useQuery({
    queryKey: ['projects', filter],
    queryFn: async () => {
      const response = await axios.get(`/api/projects?filter=${filter}`);
      return response.data.projects;
    },
    refetchOnWindowFocus: false
  });

  // Create project mutation
  const createProjectMutation = useMutation({
    mutationFn: async (projectData) => {
      const response = await axios.post('/api/projects', projectData);
      return response.data;
    },
    onSuccess: (data) => {
      queryClient.invalidateQueries({ queryKey: ['projects'] });
      setShowCreateModal(false);
      toast.success('Project created successfully!');
    },
    onError: (error) => {
      toast.error(error.response?.data?.error || 'Failed to create project');
    }
  });

  // Delete project mutation
  const deleteProjectMutation = useMutation({
    mutationFn: async (projectId) => {
      await axios.delete(`/api/projects/${projectId}`);
    },
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['projects'] });
      toast.success('Project deleted successfully!');
    },
    onError: (error) => {
      toast.error(error.response?.data?.error || 'Failed to delete project');
    }
  });

  const handleCreateProject = (projectData) => {
    createProjectMutation.mutate(projectData);
  };

  const handleDeleteProject = (projectId) => {
    if (window.confirm('Are you sure you want to delete this project?')) {
      deleteProjectMutation.mutate(projectId);
    }
  };

  if (error) {
    return (
      <DashboardContainer>
        <div>Error loading projects: {error.message}</div>
      </DashboardContainer>
    );
  }

  return (
    <DashboardContainer>
      <Header>
        <Title>My Projects</Title>
        <CreateButton onClick={() => setShowCreateModal(true)}>
          Create New Project
        </CreateButton>
      </Header>

      <FilterContainer>
        {['all', 'active', 'completed', 'archived'].map((filterOption) => (
          <FilterButton
            key={filterOption}
            active={filter === filterOption}
            onClick={() => setFilter(filterOption)}
          >
```

```
            {filterOption.charAt(0).toUpperCase() + filterOption.slice(1)}
          </FilterButton>
        ))}
      </FilterContainer>

      {isLoading ? (
        <div>Loading projects...</div>
      ) : (
        <ProjectGrid>
          {projects.map((project) => (
            <ProjectCard
              key={project._id}
              project={project}
              onDelete={() => handleDeleteProject(project._id)}
            />
          ))}
          {projects.length === 0 && (
            <div>No projects found. Create your first project!</div>
          )}
        </ProjectGrid>
      )}

      {showCreateModal && (
        <CreateProjectModal
          onSubmit={handleCreateProject}
          onClose={() => setShowCreateModal(false)}
          loading={createProjectMutation.isPending}
        />
      )}
    </DashboardContainer>
  );
};

export default ProjectDashboard;
```

Task Management with Drag and Drop:

```
 // components/TaskBoard.js
import React, { useState } from 'react';
import { DragDropContext, Droppable, Draggable } from 'react-beautiful-dnd';
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';
import axios from 'axios';
import styled from 'styled-components';
import TaskCard from './TaskCard';
import CreateTaskModal from './CreateTaskModal';

const BoardContainer = styled.div`
  display: flex;
  gap: 1rem;
  padding: 1rem;
  min-height: calc(100vh - 200px);
  overflow-x: auto;
```

```
`;

const Column = styled.div`
  background: #f8f9fa;
  border-radius: 8px;
  padding: 1rem;
  min-width: 300px;
  flex-shrink: 0;
`;

const ColumnTitle = styled.h3`
  color: #495057;
  margin-bottom: 1rem;
  font-size: 1.1rem;
  font-weight: 600;
`;

const TaskList = styled.div`
  min-height: 200px;
`;

const AddTaskButton = styled.button`
  width: 100%;
  padding: 0.75rem;
  border: 2px dashed #dee2e6;
  background: transparent;
  border-radius: 6px;
  color: #6c757d;
  cursor: pointer;
  margin-top: 1rem;
  transition: all 0.2s ease;

  &:hover {
    border-color: #667eea;
    color: #667eea;
  }
`;

const TaskBoard = ({ projectId }) => {
  const [showCreateModal, setShowCreateModal] = useState(false);
  const [selectedColumn, setSelectedColumn] = useState(null);
  const queryClient = useQueryClient();

  const columns = [
    { id: 'todo', title: 'To Do', color: '#6c757d' },
    { id: 'in-progress', title: 'In Progress', color: '#007bff' },
    { id: 'review', title: 'Review', color: '#ffc107' },
    { id: 'done', title: 'Done', color: '#28a745' }
  ];

  // Fetch tasks
  const { data: tasks = [], isLoading } = useQuery({
    queryKey: ['tasks', projectId],
    queryFn: async () => {
      const response = await axios.get(`/api/projects/${projectId}/tasks`);
      return response.data.tasks;
    }
  });

  // Update task status mutation
  const updateTaskMutation = useMutation({
    mutationFn: async ({ taskId, updates }) => {
      const response = await axios.put(`/api/tasks/${taskId}`, updates);
      return response.data;
    },
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['tasks', projectId] });
    }
  });

  const handleDragEnd = (result) => {
    if (!result.destination) return;

    const { draggableId, destination } = result;
    const newStatus = destination.droppableId;

    updateTaskMutation.mutate({
      taskId: draggableId,
      updates: { status: newStatus }
    });
  };

  const getTasksByStatus = (status) => {
    return tasks.filter(task => task.status === status);
  };

  const handleAddTask = (columnId) => {
    setSelectedColumn(columnId);
    setShowCreateModal(true);
  };

  return (
    <>
      <DragDropContext onDragEnd={handleDragEnd}>
        <BoardContainer>
          {columns.map((column) => (
            <Column key={column.id}>
              <ColumnTitle style={{ color: column.color }}>
                {column.title} ({getTasksByStatus(column.id).length})
              </ColumnTitle>

              <Droppable droppableId={column.id}>
                {(provided, snapshot) => (
                  <TaskList
                    ref={provided.innerRef}
                    {...provided.droppableProps}
                    style={{
                      backgroundColor: snapshot.isDraggingOver
```

```
                        ? 'rgba(102, 126, 234, 0.1)'
                        : 'transparent'
                    }}
                  >
                    {getTasksByStatus(column.id).map((task, index) => (
                      <Draggable
                        key={task._id}
                        draggableId={task._id}
                        index={index}
                      >
                        {(provided, snapshot) => (
                          <div
                            ref={provided.innerRef}
                            {...provided.draggableProps}
                            {...provided.dragHandleProps}
                            style={{
                              ...provided.draggableProps.style,
                              transform: snapshot.isDragging
                                ? `${provided.draggableProps.style?.transform} rotate(
                                : provided.draggableProps.style?.transform
                            }}
                          >
                            <TaskCard task={task} />
                          </div>
                        )}
                      </Draggable>
                    ))}
                    {provided.placeholder}
                  </TaskList>
                )}
              </Droppable>

              <AddTaskButton onClick={() => handleAddTask(column.id)}>
                + Add a task
              </AddTaskButton>
            </Column>
          ))}
        </BoardContainer>
      </DragDropContext>

      {showCreateModal && (
        <CreateTaskModal
          projectId={projectId}
          initialStatus={selectedColumn}
          onClose={() => {
            setShowCreateModal(false);
            setSelectedColumn(null);
          }}
        />
      )}
    </>
  );
};
```

```
export default TaskBoard;
```

Subscription and Payment Integration

Stripe Payment Setup:

```
 // server/controllers/subscriptionController.js
const stripe = require('stripe')(process.env.STRIPE_SECRET_KEY);
const User = require('../models/User');

const createSubscription = async (req, res) => {
  try {
    const { priceId, paymentMethodId } = req.body;
    const userId = req.user._id;

    // Create or retrieve Stripe customer
    let customer;
    const user = await User.findById(userId);

    if (user.subscription.stripeCustomerId) {
      customer = await stripe.customers.retrieve(user.subscription.stripeCustomerId);
    } else {
      customer = await stripe.customers.create({
        email: user.email,
        name: user.username,
        metadata: { userId: userId.toString() }
      });

      user.subscription.stripeCustomerId = customer.id;
      await user.save();
    }

    // Attach payment method to customer
    await stripe.paymentMethods.attach(paymentMethodId, {
      customer: customer.id,
    });

    // Set as default payment method
    await stripe.customers.update(customer.id, {
      invoice_settings: {
        default_payment_method: paymentMethodId,
      },
    });

    // Create subscription
    const subscription = await stripe.subscriptions.create({
      customer: customer.id,
      items: [{ price: priceId }],
      default_payment_method: paymentMethodId,
      expand: ['latest_invoice.payment_intent'],
    });

    // Update user subscription details
    user.subscription.stripeSubscriptionId = subscription.id;
    user.subscription.status = 'active';
    user.subscription.plan = getPlanFromPriceId(priceId);
    user.subscription.currentPeriodEnd = new Date(subscription.current_period_end * 10
    await user.save();

    res.json({
      subscriptionId: subscription.id,
      clientSecret: subscription.latest_invoice.payment_intent.client_secret,
      status: subscription.status
    });

  } catch (error) {
    console.error('Subscription creation error:', error);
    res.status(500).json({ error: error.message });
  }
};

// Webhook handler for Stripe events
const handleWebhook = async (req, res) => {
  const sig = req.headers['stripe-signature'];
  let event;

  try {
    event = stripe.webhooks.constructEvent(
      req.body,
      sig,
      process.env.STRIPE_WEBHOOK_SECRET
    );
  } catch (err) {
    console.error('Webhook signature verification failed:', err.message);
    return res.status(400).send(`Webhook Error: ${err.message}`);
  }

  // Handle the event
  switch (event.type) {
    case 'invoice.payment_succeeded':
      await handleSuccessfulPayment(event.data.object);
      break;

    case 'invoice.payment_failed':
      await handleFailedPayment(event.data.object);
      break;

    case 'customer.subscription.updated':
      await handleSubscriptionUpdate(event.data.object);
      break;

    case 'customer.subscription.deleted':
      await handleSubscriptionCancellation(event.data.object);
      break;

    default:
      console.log(`Unhandled event type ${event.type}`);
  }

  res.json({ received: true });
};

module.exports = {
  createSubscription,
  handleWebhook
};
```

This comprehensive project demonstrates real-world MERN stack development with production-ready features including authentication, authorization, subscription payments, real-time updates, and scalable architecture patterns.

Continue with the next two projects: E-commerce Platform and Real-Time Chat Application, plus deployment strategies for complete full-stack mastery.