1. Task Management App

Creating a Task Management App using the MEAN stack (MongoDB, Express.js, Angular, Node.js) with user authentication and real-time updates is a comprehensive project. I'll provide you with a high-level overview and code snippets to get you started. You can then expand and customize the application as needed.

- ### Backend (Node.js & Express.js)

Setting up the project structure:

Create a new Node.js project and set up your project structure. You can use a tool like Express Generator to scaffold a basic project structure.

Install necessary packages:

Install the required Node.js packages, including Express.js, Mongoose (for MongoDB interaction), Passport.js (for authentication), and Socket.io (for real-time updates).

```
npm install express mongoose passport passport-local socket.io
```

Create MongoDB schema:

Define a MongoDB schema for tasks and users. Create models for tasks and users using Mongoose.

- **javascript**

```javascript
// models/task.js
const mongoose = require('mongoose');

const taskSchema = new mongoose.Schema({
  title: String,
  description: String,
  completed: Boolean,
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
  },
});

module.exports = mongoose.model('Task', taskSchema);
```

Create a similar schema for user management.

Authentication with Passport.js:

1. Make a Task Management App

Implement user authentication using Passport.js. Set up strategies for local authentication (username and password) and JWT (JSON Web Tokens) for API authentication.

- **javascript**

```
// Passport.js configuration
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;

passport.use(
  new LocalStrategy(function (username, password, done) {
    // Verify user credentials here and call done accordingly.
  })
);
```

API Routes:

Create API routes for tasks (CRUD operations) and user authentication. Use Express.js to define these routes.

- **javascript**

```
// routes/tasks.js
const express = require('express');
const router = express.Router();

const Task = require('../models/task');

// Define your CRUD routes for tasks here.

module.exports = router;
```

Create similar routes for user authentication.

Real-time updates with Socket.io:

Implement real-time updates for tasks using Socket.io. Emit events when a new task is created or an existing task is updated or deleted.

- **javascript**

```
// Socket.io configuration

const io = require('socket.io')(server);

io.on('connection', (socket) => {

  console.log('A user connected');

  socket.on('disconnect', () => {

    console.log('A user disconnected');
  });
});
```

- Frontend (Angular)

Setting up the Angular project:

Create a new Angular project using Angular CLI and set up your project structure.

```
ng new task-manager
```

Create Angular components:

Create Angular components for task management, user authentication, and real-time updates.

```
ng generate component task-list
ng generate component task-create
ng generate component task-detail
ng generate component user-login
ng generate component user-register
```

Routing:

Set up Angular routing to navigate between different components.

- **javascript**

```
// app-routing.module.ts
const routes: Routes = [
  { path: '', redirectTo: '/tasks', pathMatch: 'full' },
  { path: 'tasks', component: TaskListComponent },
  { path: 'tasks/create', component: TaskCreateComponent },
  // Define routes for other components
];
```

Implement CRUD operations:

Implement CRUD operations for tasks in Angular components. Use Angular forms to create and edit tasks, and use Angular services to interact with the Node.js backend API.

User Authentication:

Implement user authentication in Angular components using Angular forms and services. Communicate with the Node.js backend for user registration and login.

Real-time updates with Socket.io:

Use Socket.io in your Angular components to listen for real-time updates from the Node.js server and update the task list in real-time.

MongoDB

Database Setup:

Set up a MongoDB database, and configure your Node.js application to connect to it using Mongoose.

Data Model:

Create collections for tasks and users in MongoDB to store data.

Putting It All Together

Integrate the frontend and backend by making API requests from Angular components to Node.js routes. Ensure that you handle user authentication, real-time updates, and task management properly.

This is a high-level overview of creating a Task Management App using the MEAN stack. Depending on your requirements, you can expand the features, add user roles, enhance security, and improve the user interface. Remember to handle errors gracefully and thoroughly test your application as you build it.