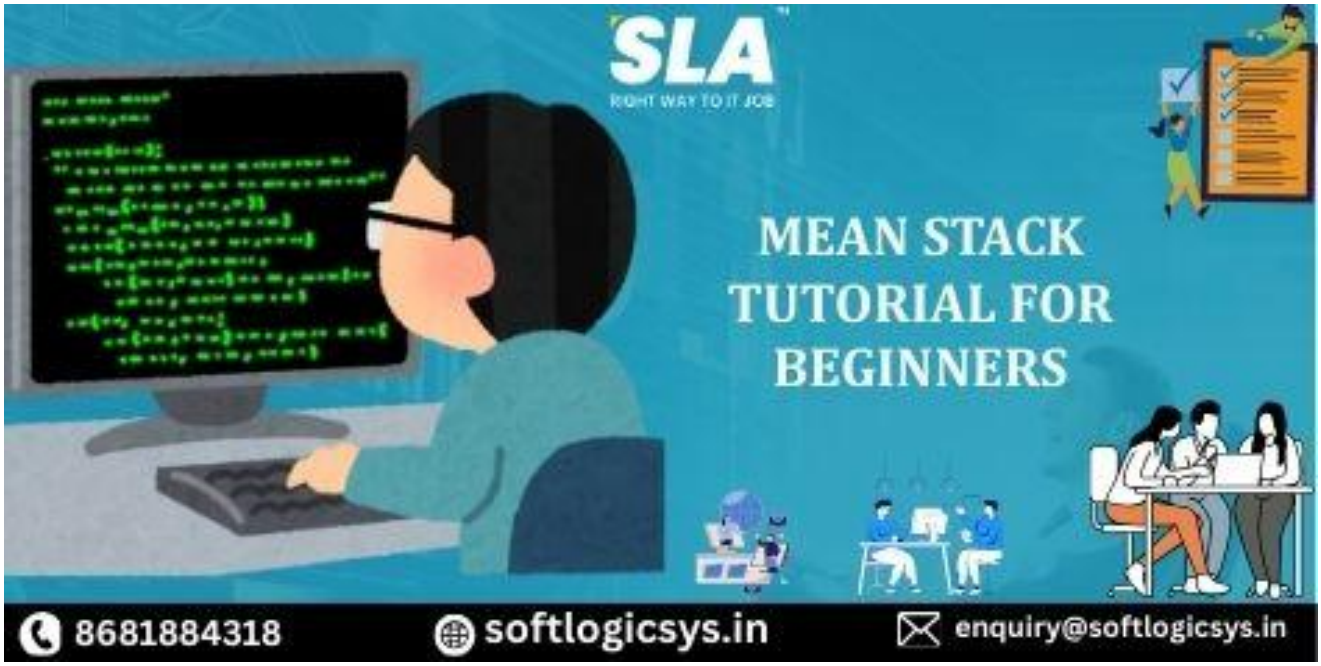

MEAN Stack Tutorial for Beginners

Introduction

Web development can seem daunting with so many frameworks and technologies to learn. The MEAN Stack (MongoDB, Express.js, Angular, Node.js) streamlines this process by offering a single, JavaScript-based solution for full-stack development. Goodbye context-switching and complicated setups! This MEAN Stack tutorial for beginners distills the complexity, presenting you with a direct path to creating robust, up-to-date applications. Ready to begin? Take a look at our **MEAN Stack course syllabus** for further learning.

Why Students or Freshers Learn MEAN Stack Development

MEAN Stack is very popular among students and freshers due to the fact that it provides a great career edge. Following are the main reasons for learning it:

- **Full-Stack Skillset:** You learn front-end and back-end development using one set of skills.

- **Jobs in Demand:** Organizations use MEAN extensively, resulting in high job demand and good pay rates.
- **JavaScript Everywhere:** Applying JavaScript to the entire application minimizes complexity and simplifies the development process.
- **Rapid Development:** MongoDB and Node.js allow fast, scalable applications to be developed quickly.
- **Community Support:** Angular and Node.js boast enormous, thriving communities for learning and debugging.

Ready for your next tech interview? Download our Top **MEAN Stack Interview Questions and Answers**!

Step-by-Step MEAN Stack Tutorial for Beginners

Beginning your journey with the MEAN Stack is an exhilarating venture into full-stack development! This step-by-step MEAN stack tutorial offers a concise explanation of how to set up your environment, create a simple application, and get familiarized with the integration of MongoDB, Express.js, Angular, and Node.js.

Step 1: Getting the Development Environment Set Up

The MEAN Stack is built on Node.js and its package manager, npm.

1.1 Install Node.js and npm

You must have Node.js installed in order to execute the server (Express/Node.js) and Angular (which relies on Node.js for its CLI).

- **Download:** Go to the official Node.js website and download the LTS (Long-Term Support) version for your platform.
- **Install:** Execute the installer and accept the default prompts. This will install Node.js and npm (Node Package Manager) automatically.
- **Verify:** Open your terminal (Command Prompt, PowerShell, or Bash) and execute the following commands to verify successful installation:

node -v

npm -v

You should see installed version numbers.

1.2 Installing MongoDB

MongoDB is the NoSQL database for the MERN Stack.

- **Download:** Navigate to the MongoDB Community Server download page and download the installer for your OS.
- **Install:** Execute the installer. On Windows, be sure to choose the “Install MongoDB Compass” option since Compass is an excellent GUI utility for your database.
- **Start MongoDB:** MongoDB runs normally as a background service. You can verify its state in your system services manager. The default connection URL will be *mongodb://localhost:27017*.

1.3 Install Angular CLI

The Angular Command Line Interface (CLI) is used for scaffolding, building, and developing Angular applications.

Install the Angular CLI globally with npm:

npm install -g @angular/cli

Step 2: Creating the MEAN Project Structure

A standard MEAN app keeps the server (backend) and the client (frontend) separate.

2.1 Create the Main Project Folder

Open your terminal and create a directory for your project:

mkdir mean-todo-app

cd mean-todo-app

2.2 Initialize the Angular Client

Create the frontend Angular app within the main directory:

ng new client-app –no-standalone

- **ng new client-app:** Creates a new Angular project called client-app.
- **–no-standalone:** (Angular 17+ onwards) Beginners use this flag to maintain the conventional module-based architecture, prevalent in older tutorials, but you can skip it and use the default standalone components if you like.

We will go ahead assuming the conventional module structure for this tutorial's sake.

When prompted:

- **Routing:** Enter y for Angular routing.
- **Stylesheet format:** Select CSS (or your choice).

2.3 Initialize the Express Server

Create a directory for the backend server and initialize a Node.js project:

mkdir server

cd server

npm init -y

This command creates a package.json file in the server directory.

Step 3: Building the Backend (Express & Node.js)

3.1 Install Backend Dependencies

Install Express (the web framework), Mongoose (the MongoDB ODM), and CORS (to enable communication from the frontend).

npm install express mongoose cors dotenv

- **.dotenv:** Used to store environment variables (such as the MongoDB connection string).

3.2 Configure the Server (server/index.js)

Create the main server file, index.js, within the server directory.

server/index.js

```
// Load environment variables from .env file

require('dotenv').config();

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

const app = express();

const PORT = process.env.PORT || 3000;
```

```
// Middleware

app.use(cors()); // Enable Cross-Origin Resource Sharing

app.use(express.json()); // Parse incoming JSON requests

// MongoDB Connection

const MONGODB_URI = process.env.MONGODB_URI ||
'mongodb://localhost:27017/todoDB';

mongoose.connect(MONGODB_URI)

  .then(() => console.log('MongoDB Connected Successfully!'))

  .catch(err => console.error('MongoDB Connection Error:', err));

// Basic Route for testing

app.get('/', (req, res) => {

  res.send('MEAN Stack Server is Running!');

});

// Start the server

app.listen(PORT, () => {

  console.log(`Server is running on port ${PORT}`);

});
```

3.3 Create the Todo Model (server/models/Todo.js)

Create a models directory within server. This establishes the structure (schema) for your Todo items within MongoDB.

server/models/Todo.js

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

const Schema = mongoose.Schema;

// Define the schema for a Todo item

const TodoSchema = new Schema({

  text: {

    type: String,

    required: true,

    trim: true

  },

  completed: {

    type: Boolean,

    default: false

  },

  createdAt: {
```

```
    type: Date,
    default: Date.now
  }
});

module.exports = mongoose.model('Todo', TodoSchema);
```

3.4 Create the API Routes (server/routes/todoRoutes.js)

Create a routes directory within server. This file is responsible for the logic of creating, reading, updating, and deleting (CRUD) todos.

server/routes/todoRoutes.js

```
const express = require('express');
const router = express.Router();
const Todo = require('../models/Todo');
// GET all todos
router.get('/', async (req, res) => {
  try {
    const todos = await Todo.find().sort({ createdAt: -1 });
    res.json(todos);
  } catch (err) {
```

```
    res.status(500).json({ message: err.message });
  }
});
// POST a new todo
router.post('/', async (req, res) => {
  const todo = new Todo({
    text: req.body.text
  });
  try {
    const newTodo = await todo.save();
    res.status(201).json(newTodo);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
});
// PUT (Update) a todo's completed status
router.put('/:id', async (req, res) => {
```

```
  try {
    const todo = await Todo.findById(req.params.id);
    if (!todo) {
      return res.status(404).json({ message: 'Todo not found' });
    }
    // Toggle the completed status
    todo.completed = !todo.completed;
    const updatedTodo = await todo.save();
    res.json(updatedTodo);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
});
// DELETE a todo
router.delete('/:id', async (req, res) => {
  try {
    const todo = await Todo.findByIdAndDelete(req.params.id);
```

```
    if (!todo) {
      return res.status(404).json({ message: 'Todo not found' });
    }
    res.json({ message: 'Todo deleted successfully' });
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});
module.exports = router;
```

3.5 Integrate Routes into Server

Return to server/index.js and include the Todo routes prior to app.listen.

server/index.js (Updated Snippet)

```
// … (imports and middleware)
// MongoDB Connection
// … (mongoose.connect)
// Import Todo Routes
const todoRoutes = require('./routes/todoRoutes');
// Use Todo Routes – All routes will be prefixed with '/api/todos'
```

```
app.use('/api/todos', todoRoutes);

// Basic Route for testing (can be removed later)

app.get('/', (req, res) => {

  res.send('MEAN Stack Server is Running!');

});

// Start the server

// … (app.listen)
```

3.6 Run the Server

Update your server/package.json start script for easy execution.

```
"scripts": {

  "start": "node index.js",

  "test": "echo \"Error: no test specified\" && exit 1"

},
```

Run the server from the server directory now:

```
node index.js
```

You should notice: "MongoDB Connected Successfully!" and "Server is running on port 3000".

Step 4: Building the Frontend (Angular)

The Angular client consume the REST API created by the express server.

4.1 Create the Todo Service

Go to the client-app directory and create a service to make API calls.

cd ../client-app

ng generate service todo

client-app/src/app/todo.service.ts

```
import { Injectable } from '@angular/core';

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

// Define a simple interface for our Todo object

interface Todo {

  _id: string;

  text: string;

  completed: boolean;

}

@Injectable({

  providedIn: 'root'

})
```

```
export class TodoService {
  private apiUrl = 'http://localhost:3000/api/todos'; // Connects to Express API
  constructor(private http: HttpClient) { }
  // GET: Fetch all todos
  getTodos(): Observable<Todo[]> {
    return this.http.get<Todo[]>(this.apiUrl);
  }
  // POST: Create a new todo
  addTodo(text: string): Observable<Todo> {
    return this.http.post<Todo>(this.apiUrl, { text });
  }
  // PUT: Update todo completed status
  toggleCompleted(id: string, completed: boolean): Observable<Todo> {
    return this.http.put<Todo>(`${this.apiUrl}/${id}`, { completed });
  }
  // DELETE: Remove a todo
  deleteTodo(id: string): Observable<any> {
```

```
    return this.http.delete(`${this.apiUrl}/${id}`);
  }
}
```

4.2 Import HttpClientModule

In Angular, you must import the *HttpClientModule* in order to use the HttpClient.

Open *client-app/src/app/app.module.ts* and include it in the imports array.

client-app/src/app/app.module.ts (Snippet – Standard module setup)

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http'; // <– Import this
import { FormsModule } from '@angular/forms'; // <– Import this for forms
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
```

```
    BrowserModule,
    AppRoutingModule,
    HttpClientModule, // <– Add here
    FormsModule // <– Add here
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

4.3 Implement the Component Logic

Modify the main component to utilize the service and control the todo list state.

client-app/src/app/app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { TodoService } from './todo.service';
interface Todo {
  _id: string;
  text: string;
  completed: boolean;
```

```
}

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  title = 'MEAN Todo App';
  todos: Todo[] = [];
  newTodoText: string = '';
  constructor(private todoService: TodoService) {}
  ngOnInit(): void {
    this.fetchTodos();
  }
  // Retrieve todos from the backend
  fetchTodos(): void {
    this.todoService.getTodos().subscribe({
```

```
    next: (data) => this.todos = data,
    error: (err) => console.error('Error fetching todos:', err)
  });
}
// Add a new todo
addTodo(): void {
  if (!this.newTodoText.trim()) return;
  this.todoService.addTodo(this.newTodoText).subscribe({
    next: (newTodo) => {
      this.todos.unshift(newTodo); // Add new todo to the start of the array
      this.newTodoText = ''; // Clear input field
    },
    error: (err) => console.error('Error adding todo:', err)
  });
}
// Toggle todo completion status
toggleCompleted(todo: Todo): void {
```

```
  const updatedCompleted = !todo.completed;
  this.todoService.toggleCompleted(todo._id, updatedCompleted).subscribe({
    next: (updatedTodo) => {
      // Find the index of the updated todo and replace it
      const index = this.todos.findIndex(t => t._id === updatedTodo._id);
      if (index > -1) {
        this.todos[index] = updatedTodo;
      }
    },
    error: (err) => console.error('Error toggling todo:', err)
  });
}
// Delete a todo
deleteTodo(id: string): void {
  this.todoService.deleteTodo(id).subscribe({
    next: () => {
      this.todos = this.todos.filter(t => t._id !== id); // Remove from local array
```

```
      },
      error: (err) => console.error('Error deleting todo:', err)
    });
  }
}
```

4.4 Create the Component Template (HTML)

Replace the contents of client-app/src/app/app.component.html with the following:

client-app/src/app/app.component.html

```
<div class="container">
  <h1>{{ title }}</h1>
  <form (ngSubmit)="addTodo()">
    <input
      type="text"
      placeholder="Add a new task…"
      [(ngModel)]="newTodoText"
      name="newTodo"
      required
    >
```

```
    <button type="submit">Add Task</button>
  </form>
  <ul class="todo-list">
    <li *ngFor="let todo of todos" [class.completed]="todo.completed">
      <span (click)="toggleCompleted(todo)">
        {{ todo.text }}
      </span>
      <button class="delete-btn" (click)="deleteTodo(todo._id)">
        X
      </button>
    </li>
  </ul>
  <p *ngIf="todos.length === 0" class="no-todos">No tasks yet! Time to add one.</p>
</div>
```

4.5 Add Basic Styling (CSS)

Add some simple styles to client-app/src/app/app.component.css to make it presentable.

client-app/src/app/app.component.css

```
.container {
  max-width: 600px;
  margin: 50px auto;
  padding: 20px;
  font-family: Arial, sans-serif;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  border-radius: 8px;
}
h1 {
  text-align: center;
  color: #333;
}
/* Form Styling */
form {
  display: flex;
  margin-bottom: 20px;
}
```

```
input[type="text"] {
  flex-grow: 1;
  padding: 10px;
  border: 1px solid #ddd;
  border-radius: 4px 0 0 4px;
  font-size: 16px;
}
button[type="submit"] {
  padding: 10px 15px;
  background-color: #5cb85c;
  color: white;
  border: none;
  border-radius: 0 4px 4px 0;
  cursor: pointer;
}
button[type="submit"]:hover {
  background-color: #4cae4c;
```

```
}

/* List Styling */

.todo-list {

  list-style: none;

  padding: 0;

}

.todo-list li {

  display: flex;

  justify-content: space-between;

  align-items: center;

  padding: 10px;

  margin-bottom: 8px;

  background-color: #f9f9f9;

  border-left: 5px solid #007bff;

  border-radius: 4px;

  transition: background-color 0.2s;

}
```

```
.todo-list li span {
  flex-grow: 1;
  cursor: pointer;
  padding: 0 10px;
}
.todo-list li.completed {
  background-color: #e0f7fa;
  border-left-color: #28a745;
}
.todo-list li.completed span {
  text-decoration: line-through;
  color: #888;
}
.delete-btn {
  background-color: #dc3545;
  color: white;
  border: none;
```

```
  padding: 5px 10px;
  border-radius: 4px;
  cursor: pointer;
  font-weight: bold;
}
.delete-btn:hover {
  background-color: #c82333;
}
.no-todos {
  text-align: center;
  color: #666;
  margin-top: 20px;
  font-style: italic;
}
```

Step 5: Running the Full Stack

You require two terminals running simultaneously to run the full MEAN Stack application: one for the backend and another for the frontend.

5.1 Start the Backend

Ensure you are in the server directory.

cd mean-todo-app/server

node index.js

(Ensure you observe “MongoDB Connected Successfully!” and “Server is running on port 3000”)

5.2 Start the Frontend (Angular)

Open a new terminal, go to the client-app directory.

cd mean-todo-app/client-app

ng serve –open

The Angular development server will compile the client application and automatically open your web browser to http://localhost:4200/.

Testing the Application

- **Add:** Enter a task into the input box and press “Add Task.” The Angular component invokes the Express POST route, which stores the task in MongoDB.
- **View:** The new task is immeiately added to the list (due to the unshift logic), fetched from the Express GET route.
- **Toggle:** Click on the task text. The Angular component invokes the Express PUT route, which marks the completed status in MongoDB.
- **Delete:** Click the ‘X’ button. The Angular component invokes the Express DELETE route, which deletes the item from MongoDB.

You have now successfully developed and executed your first complete MEAN application. This project gives you a good start to learning the flow of communication among the four fundamental technologies.

Real Time Examples for MEAN Stack Tutorial for Learners

MEAN Stack is employed to develop high-performance, elastic applications in numerous industries. Looking at real-life examples reinforces your learning:

- **Social Networking Sites:** Development of chat functionality, customized news feeds, and live update notification systems (Node.js/Express for performance, MongoDB for schema-less data, Angular for dynamic UI).
- **E-commerce Websites:** Managing product catalogs, shopping carts, and authentication of users with dynamic updates without complete page reloads.
- **Content Management Systems (CMS):** Developing blogging platforms or administration dashboards in which content updates in real time (Angular takes care of administrative UI, Node/Express handles content serving).
- **Real-Time Collaborative Tools:** Developing applications such as Trello or collaborative document editors (Node.js is great with WebSockets for real-time data synchronization).
- **Data Streaming and Analytics:** Processing streams of user activity data and rendering real-time analytical graphs on the frontend.

Ready to apply your skills? Explore our **MEAN Stack project ideas**.

FAQs About MEAN Stack Developer Tutorial for Beginners

1. Is MEAN stack good for beginners?

Yes, MEAN Stack is suitable for beginners, particularly for those with a JavaScript background. Having one language throughout the entire stack (MongoDB, Express, Angular, Node.js) gives a unified, integrated learning experience, making the jump from frontend to backend easy.

2. Is MEAN or MERN better?

Neither is always “better”; it all depends on the project. MERN (React) is usually favored for quick development and versatility, and MEAN (Angular) is generally used for complex, large, and enterprise-level applications because of Angular’s opinionated, structured framework.

3. What do you mean by MEAN stack?

MEAN Stack is an abbreviation for a collection of JavaScript technologies utilized for full-stack web development: MongoDB (database), Express.js (backend framework), Angular (frontend framework), and Node.js (runtime environment).

4. Can I learn JS in 3 days?

No, it is not possible to get job-ready skills or become proficient in JavaScript (JS) within 3 days. You can learn the syntax and fundamental concepts within those 3 days, but to master it and use it for advanced web development typically requires a few months of regular practice.

5. Is web dev a stressful job?

Web development can be stressful because of looming deadlines, pressure of ongoing learning with ever-changing technologies, dealing with intricate client expectations, and debugging stubborn technical problems.

6. Can I learn React js in 7 days?

No, learning React.js in 7 days is unrealistic. You can grasp the basic concepts (like components and JSX) in a week if you already know JavaScript well, but building functional, state-managed applications requires several weeks or months of dedicated practice.

7. Is MERN stack dead in 2025?

No, the MERN Stack is far from dead in 2025. It is still extremely popular and sought after for developing modern, scalable, and highly interactive web applications because of the dominance by React and the entire JavaScript ecosystem.

8. Which is best, Django or MERN?

Django (Python backend) or MERN (JavaScript full stack) is a matter of requirements. MERN tends to be more suitable for real-time single-page applications (SPAs), whereas Django is superb for intricate, solid, data-driven applications that require high security and quick backend feature implementation.

9. Is MEAN stack frontend or backend?

MEAN Stack is neither all backend nor frontend; it's a full-stack solution. Angular takes care of the frontend (client-side), and MongoDB, Express.js, and Node.js together take care of the backend (server-side and database).

10. How to run MEAN project?

In order to execute a MEAN project, you generally have two different terminals: one to execute the backend server (Express.js/Node.js) and one to execute the frontend application (Angular). You initially go into the corresponding folders and execute their start commands (node index.js and ng serve).

11. What is the salary of MEAN stack developer?

The **MEAN Stack Developer salary** is quite different based on location, experience, and size of company. In India, an average salary would be around ₹12 – ₹15 lakhs per year but can go much higher for experienced ones in large tech cities.

12. Is MEAN stack still in demand?

Yes, the MEAN Stack remains popular, especially in companies that appreciate Angular's disciplined architecture for large-scale, maintainable applications. Its complete JavaScript environment guarantees its ongoing popularity in the full-stack development space.

Conclusion

Congratulations on completing this introductory MEAN Stack tutorial, relating MongoDB, Express.js, Angular, and Node.js to create a complete application stack. You are familiar with the strength of having a single, JavaScript-based environment for quick development. What follows next is the mastery of advanced topics such as

authentication, state management, and deployment to become a ready-for-hire developer.

Want to convert your foundational skills to professional skills? Join our Complete **MEAN Stack Developer Course in Chennai**.