9. URL Shortened

Creating a URL Shortened service using the MEAN stack (MongoDB, Express.js, Angular, Node.js) is an interesting project. It involves both a front-end for generating short URLs and a back-end for redirecting and tracking click statistics. Here's a high-level overview and some code snippets to get you started:

Project Setup and Structure

Set up a new project folder and structure for your URL Shortener. Install the required Node.js packages and create a basic Angular application.

```
# Create a new Angular application
ng new url-shortener
```

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

Create the backend of your URL Shortener service using Node.js and Express.js.

Installation of Packages

Install the necessary packages for Express.js, Mongoose, and other dependencies.

```
npm install express mongoose cors
```

Setting up Express.js

Create your Express.js server, set up middleware, and handle routes.

- **javascript**

```
// server.js

const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const shortid = require('shortid'); // for generating short URLs

const app = express();

// Middleware
app.use(express.json());
app.use(cors());

// Database connection
mongoose.connect('mongodb://localhost/url-shortener', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
```

```
  useCreateIndex: true,
});

// Define a Mongoose model for URL data
const Url = mongoose.model('Url', {
  originalUrl: String,
  shortUrl: String,
  clickCount: Number,
});

// Routes
app.post('/api/shorten', async (req, res) => {
  const { originalUrl } = req.body;

  // Generate a unique short ID
  const shortUrl = shortid.generate();

  // Create a new URL document
  const url = new Url({
    originalUrl,
    shortUrl,
```

```
    clickCount: 0,
  });

  // Save the URL to the database
  await url.save();

  res.json({ shortUrl });
});

app.get('/:shortUrl', async (req, res) => {
  const { shortUrl } = req.params;

  // Find the original URL in the database
  const url = await Url.findOne ({ shortUrl });

  if (!url) {
    return res.status(404).json({ error: 'Short URL not found' });
  }

  // Increment the click count
  url.clickCount++;
```

```
  await url.save();

  // Redirect to the original URL
  res.redirect(url.originalUrl);
});

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

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

- ## Frontend (Angular)

Create the frontend of your URL Shortener using Angular. Design the user interface for shortening URLs and display the shortened URL.

Design and UI

Design the user interface for your URL Shortener using Angular components, templates, and styles.

URL Shortening

Create components and forms for users to input long URLs and get back shortened URLs.

- **typescript**

```
// url-shorten.component.ts
import { Component } from '@angular/core';
import { UrlService } from './url.service';

@Component({
  selector: 'app-url-shorten',
  templateUrl: './url-shorten.component.html',
})
export class UrlShortenComponent {
  originalUrl: string;
  shortUrl: string;

  constructor(private urlService: UrlService) {}

  shortenUrl() {
    this.urlService.shortenUrl(this.originalUrl).subscribe((data) => {
      this.shortUrl = data.shortUrl;
```

```
    });
  }
}
```

MongoDB

Create a MongoDB database to store original and shortened URL data along with click statistics.

Putting It All Together

Integrate the frontend and backend by making API requests from Angular components to Node.js routes. Ensure that you handle URL shortening, redirection, and tracking click statistics properly.

Building a URL Shortener is a practical and educational project, and it can be expanded with additional features such as user accounts, analytics, and more advanced tracking.