URL Shortener using Node.js, Express, and MongoDB:

We now need to download the packages.

Run the following command on your terminal to download them:

```
npm install express mongoose shortid valid-url
```

This will download the named packages inside node_modules folder and update the package.json file with the dependencies.

The npm packages will include:

- express: Express is a backend web application framework for Node.js used for building web applications and APIs.
- mongoose: Mongoose is an asynchronous database driver or Object Data Mapper for MongoDB. It is responsible for connecting to the database and performing query operations.
- short-id: The short-id module creates user-friendly and unique ids for our URLs.
- valid-url: This is a module that verifies all the URLs sent to the API.
- nodemon: nodemon package will be installed as a development dependency. It will constantly monitor our applications by automatically restarting the server when any file changes.

```
npm install --save-dev nodemon.
```

1. Configuring the MongoDB connection inside the ‘config’

We will use the mongoose package that we installed via npm as the database driver. To configure the database, create a config folder inside our URL-Shortener-Service folder. Inside the config folder, add a file named config.js.

2. The database model for URL details

When using mongoose, models are defined using a Schema interface. A Schema will allow us to define all the fields stored in each document along with the validation or default values.

To create a model, we need to create a schema interface by importing the mongoose npm package. The mongoose.Schema method is instantiated to define with and object argument. This object takes the values that our MongoDB document will store.

The values include:

- The urlCode is a string property that will store the unique ID related to each URL.
- The longUrl is the default URL which we need to shorten.
- The property shortUrl is the actual short URL that will be generated.
- The date property has a default property and is created once the model is instantiated in the database.

3. Defining routes

Our routes will be on a separate folder. Inside the URL-Shortener-Service folder, create a folder named routes.

We will create two route files named:

1. PostUrl.js: This will be a POST route that takes an incoming request with the long URL and creates the short URL and inserts it into the database. Navigate in the routes folder and create a file named url.js.
2. GetUrl.js: This is a GET for our URL redirects. It takes the short URL and redirects it to the actual long URL in the browser. To add this file, create a separate file named redirect.js inside the routes folder.

Now that we have created a POST route that creates the short URL, we need to perform a redirect so that our short URL points to the actual URL. This is a GET request to our Node.js API to query.

(baseUrl variable) as http:\\localhost:5000.

4. The final ‘index.js’ entry file

To make our routes work, we have to use a middleware pattern such as Express. Middleware functions have access to request (req) and response object (res) in the application’s request-response cycle. This explains the app.use() methods.

The first middleware allows our application to parse incoming request data format in JSON format. The app.use('/', require('./routes/redirect')) is the base URI that will configure the redirect route.

In our POST route, the base URL is /api/url and the middleware as app.use('/api/url', require('./routes/url')). Next, we need to test the application in Postman.