To create a REST service in Node.js using multiple files for better structure and modularity, follow these steps. We'll use **Express.js** to handle the REST API routes.

Project Structure:

```
/my-rest-api
|
├── /controllers
|   └── userController.js
|
├── /routes
|   └── userRoutes.js
|
├── /models
|   └── userModel.js
|
├── /config
|   └── dbConfig.js
|
├── app.js
└── package.json
```

1. Initialize Project

First, initialize a new Node.js project and install the necessary dependencies:

```bash
mkdir my-rest-api
cd my-rest-api
npm init -y
npm install express mongoose body-parser
```

2. Create `app.js`

This file will be the entry point of your application. It initializes the Express app, connects to the database, and registers routes.

```javascript
// app.js
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const userRoutes = require('./routes/userRoutes');

// Create Express App
const app = express();

// Middleware
app.use(bodyParser.json());

// Connect to MongoDB (you can change the URL as needed)
mongoose.connect('mongodb://localhost:27017/mydatabase', {
```

```
  useNewUrlParser: true,
  useUnifiedTopology: true,
}).then(() => {
  console.log('Connected to the database');
}).catch(err => {
  console.log('Failed to connect to the database:', err);
});

// Register routes
app.use('/api/users', userRoutes);

// Start the server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

3. Create Routes: `userRoutes.js`

The **routes** directory will define the endpoints. Here we define a route file for handling user-related API routes.

```javascript
// routes/userRoutes.js
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');
```

```
// GET /api/users - Get all users
router.get('/', userController.getAllUsers);

// POST /api/users - Create a new user
router.post('/', userController.createUser);

// GET /api/users/:id - Get a single user by ID
router.get('/:id', userController.getUserById);

// PUT /api/users/:id - Update a user by ID
router.put('/:id', userController.updateUser);

// DELETE /api/users/:id - Delete a user by ID
router.delete('/:id', userController.deleteUser);

module.exports = router;
```

4. Create Controller: `userController.js`

The **controllers** directory will contain the business logic for the API. Each function corresponds to a route.

```javascript
// controllers/userController.js
const User = require('../models/userModel');

// Get all users
exports.getAllUsers = async (req, res) => {
```

```
  try {
    const users = await User.find();
    res.status(200).json(users);
  } catch (err) {
    res.status(500).json({ error: 'Failed to fetch users' });
  }
};

// Create a new user
exports.createUser = async (req, res) => {
  try {
    const newUser = new User(req.body);
    await newUser.save();
    res.status(201).json(newUser);
  } catch (err) {
    res.status(500).json({ error: 'Failed to create user' });
  }
};

// Get a user by ID
exports.getUserById = async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    if (!user) {
      return res.status(404).json({ error: 'User not found' });
    }
    res.status(200).json(user);
  } catch (err) {
    res.status(500).json({ error: 'Failed to fetch user' });
```

```
  }
};

// Update a user by ID
exports.updateUser = async (req, res) => {
  try {
    const updatedUser = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });
    if (!updatedUser) {
      return res.status(404).json({ error: 'User not found' });
    }
    res.status(200).json(updatedUser);
  } catch (err) {
    res.status(500).json({ error: 'Failed to update user' });
  }
};

// Delete a user by ID
exports.deleteUser = async (req, res) => {
  try {
    const deletedUser = await User.findByIdAndDelete(req.params.id);
    if (!deletedUser) {
      return res.status(404).json({ error: 'User not found' });
    }
    res.status(200).json({ message: 'User deleted' });
  } catch (err) {
    res.status(500).json({ error: 'Failed to delete user' });
  }
};
```

5. Create Model: `userModel.js`

The **models** directory will define the MongoDB schema and interact with the database.

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

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
}, { timestamps: true });

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

6. Create Configuration: `dbConfig.js` (Optional)

This file can contain MongoDB connection settings if you want to keep configuration separate.

```javascript
// config/dbConfig.js
const mongoose = require('mongoose');

const connectDB = async () => {
 try {
  await mongoose.connect('mongodb://localhost:27017/mydatabase', {
   useNewUrlParser: true,
   useUnifiedTopology: true,
  });
  console.log('Database connected');
 } catch (error) {
  console.log('Database connection failed', error);
  process.exit(1);
 }
};

module.exports = connectDB;
```

7. Test the REST API

To test the API, you can use **Postman** or **cURL**.

- **GET** `/api/users`: Fetch all users
- **POST** `/api/users`: Create a new user (with `name`, `email`, and `password` in the request body)
- **GET** `/api/users/:id`: Get a user by ID
- **PUT** `/api/users/:id`: Update a user by ID
- **DELETE** `/api/users/:id`: Delete a user by ID

Running the API

Run the server with:

```bash
node app.js
```

If everything is set up correctly, the server will start on port 3000, and you can start sending requests to the API endpoints.