**Step-by-Step Guide to Implementing Two-Factor Authentication with Node.js**

[[](https://medium.com/@adarsh-d?source=post_page-----ca6de34a6fcb--------------------------------)](https://medium.com/@adarsh-d?source=post_page-----ca6de34a6fcb--------------------------------)

[[Dev Genius](https://blog.devgenius.io/?source=post_page-----ca6de34a6fcb--------------------------------)](https://blog.devgenius.io/?source=post_page-----ca6de34a6fcb--------------------------------)

[Adarsh Dayanand](https://medium.com/@adarsh-d?source=post_page-----ca6de34a6fcb--------------------------------)

·

Follow

Published in

[Dev Genius](https://blog.devgenius.io/?source=post_page-----ca6de34a6fcb--------------------------------)

·

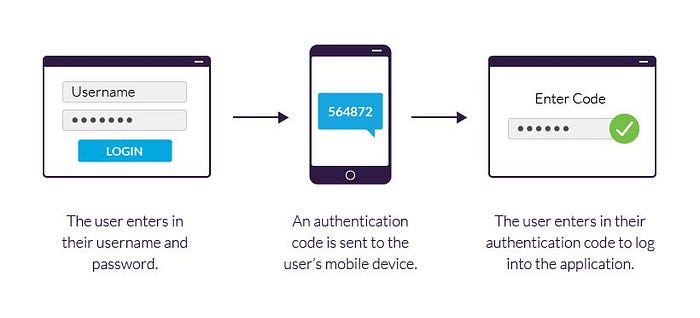
5 min read

·

Mar 23

145

1



Two-factor authentication (2FA) is a security mechanism that requires users to provide two different authentication factors to access their account: a password and a code sent to their mobile device. In this tutorial, we will walk through the steps to implement 2FA in a Node.js and Express application using the npm package speakeasy.

Step 1: Install Dependencies First, you need to install the express and speakeasy packages by running the following command in your terminal:

npm install express speakeasy

Step 2: Configure Express Next, create a new Express application and configure it to use JSON middleware and a static folder for serving static assets:

const express = require('express');  
const app = express();  
  
app.use(express.json());  
app.use(express.static('public'));  
app.listen(3000, () => {  
 console.log('Server started on port 3000');  
});

Step 3: Create a User Model Create a User model to store user data in a database. In this example, we will use a simple array to store user data:

const users = [];  
  
class User {  
 constructor(id, name, email, password, secret) {  
 this.id = id;  
 this.name = name;  
 this.email = email;  
 this.password = password;  
 this.secret = secret;  
 }  
}  
module.exports = { users, User };

Step 4: Create a Registration Route Create a POST route to handle user registration. In this route, we will generate a secret key for the user and save it in the database. We will also send a QR code to the user, which they can scan to add the account to their 2FA app:

const { users, User } = require('./user');  
const speakeasy = require('speakeasy');  
const QRCode = require('qrcode');  
  
app.post('/register', (req, res) => {  
 const { name, email, password } = req.body;  
 // Generate a new secret key for the user  
 const secret = speakeasy.generateSecret({ length: 20 });  
 // Save the user data in the database  
 const user = new User(users.length + 1, name, email, password, secret.base32);  
 users.push(user);  
 // Generate a QR code for the user to scan  
 QRCode.toDataURL(secret.otpauth\_url, (err, image\_data) => {  
 if (err) {  
 console.error(err);  
 return res.status(500).send('Internal Server Error');  
 }  
 // Send the QR code to the user  
 res.send({ qrCode: image\_data });  
 });  
});

Step 5: Create a Login Route Create a POST route to handle user login. In this route, we will validate the user's credentials and ask for a verification code from the user's 2FA app. We will use the speakeasy package to generate and verify the code:

const { users } = require('./user');  
const speakeasy = require('speakeasy');  
  
app.post('/login', (req, res) => {  
 const { email, password, token } = req.body;  
 // Find the user with the given email address  
 const user = users.find(u => u.email === email);  
 // Validate the user's credentials  
 if (!user || user.password !== password) {  
 return res.status(401).send('Invalid credentials');  
 }  
 // Verify the user's token  
 const verified = speakeasy.totp.verify({  
 secret: user.secret,  
 encoding: 'base32',  
 token,  
 window: 1  
 });  
 if (!verified) {  
 return res.status(401).send('Invalid token');  
 }  
 // User is authenticated  
 res.send('Login successful');  
});

In this route, we first extract the email, password, and token fields from the request body. We then find the user with the given email address in the users array.

Next, we validate the user’s credentials by checking if the user exists and if their password matches the one provided in the request.

If the user’s credentials are valid, we use the speakeasy package to verify the token. We pass in the user's secret key, the encoding (which should be base32), the token provided in the request, and a window of 1 (which means that the token will be valid for the current time and the previous time period).

If the token is invalid, we return a 401 Unauthorized status code with the message “Invalid token”.

If the token is valid, we send a 200 OK status code with the message “Login successful”. At this point, the user is authenticated and can access protected resources in your application.

Step 6: Create a middleware to verify if the user has successfully logged in.

const speakeasy = require('speakeasy');  
  
exports.requireToken = (req, res, next) => {  
 const { token } = req.body;  
 // Find the user with the given email address  
 const user = users.find(u => u.email === req.user.email);  
 // Verify the user's token  
 const verified = speakeasy.totp.verify({  
 secret: user.secret,  
 encoding: 'base32',  
 token,  
 window: 1  
 });  
 if (!verified) {  
 return res.status(401).send('Invalid token');  
 }  
 // Token is valid, proceed to the next middleware or route handler  
 next();  
}

This function takes in the req, res, and next arguments, which are standard for Express middleware functions. It first extracts the token from the request body.

It then finds the user with the email address stored in req.user.email. Note that we assume that the req.user object has been set by a previous middleware that performs authentication (e.g. Passport.js).

Next, it uses the speakeasy package to verify the token. It passes in the user's secret key, the encoding (which should be base32), the token provided in the request, and a window of 1 (which means that the token will be valid for the current time and the previous time period).

If the token is invalid, it returns a 401 Unauthorized status code with the message “Invalid token”. If the token is valid, it calls the next() function to proceed to the next middleware or route handler.

To use this middleware function on routes that require a valid 2FA token, you can simply add it as a middleware function before the route handler, like this:

app.post('/protected', requireToken, (req, res) => {  
 // This route handler will only be called if the user's token is valid  
 res.send('Protected resource accessed successfully');  
});

In conclusion, two-factor authentication (2FA) is a powerful security mechanism that adds an extra layer of protection to user accounts. By requiring users to provide two different authentication factors, such as a password and a code sent to their mobile device, 2FA can help prevent unauthorized access to sensitive information.

Implementing 2FA in a Node.js and Express application is relatively straightforward using the speakeasy package. By generating a secret key for each user and verifying a code provided by the user's 2FA app, you can ensure that only authorized users can access protected resources in your application.

While implementing 2FA does require additional development effort, the benefits of increased security and reduced risk of data breaches make it well worth the investment.