Aqui no **emailService.js** utilizamos o módulo **nodemailer** para enviar e-mails de forma programática. Ele cria um "transporter" que serve como um canal de comunicação com um serviço de e-mail (como Gmail) e uma função que envia um e-mail com base nas configurações fornecidas.

1. Importação do Módulo nodemailer:

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

- **nodemailer**: Um módulo Node.js popular para enviar e-mails. Ele oferece métodos fáceis para configurar o envio de e-mails a partir de um servidor ou serviço de e-mail.

2. Configuração do transporter:

```
const transporter = nodemailer.createTransport({
 service: 'gmail', // Use o serviço de e-mail de sua escolha
 auth: {
  user: process.env.EMAIL_USER, // Seu e-mail definido nas variáveis de ambiente
  pass: process.env.EMAIL_PASS // Sua senha de e-mail definida nas variáveis de ambiente
 }
});
```

- **nodemailer.createTransport()**: Essa função cria um **transporter**, que é o objeto que lida com o envio real dos e-mails.
- **service: 'gmail'**: Aqui, o código especifica que o serviço de e-mail usado será o **Gmail**. O nodemailer suporta vários serviços de e-mail, mas você pode mudar o serviço para outros como Outlook, Yahoo, ou qualquer servidor SMTP personalizado.
- **auth**: A autenticação necessária para acessar o serviço de e-mail. Geralmente, são necessárias credenciais de login:
    - **user**: O endereço de e-mail usado para enviar os e-mails. No código, ele está sendo acessado a partir das variáveis de ambiente (process.env.EMAIL_USER) para manter as credenciais seguras.
    - **pass**: A senha ou o token de aplicação (no caso de Gmail, tokens são mais recomendados) também é armazenado em uma variável de ambiente (process.env.EMAIL_PASS) para proteger as informações sensíveis.

3. Função sendEmail:

1. Essa função é usada para enviar um e-mail.

```
const sendEmail = (to, subject, text) => {
  const mailOptions = {
    from: process.env.EMAIL_USER,
    to,
    subject,
    text
  };
```

- **mailOptions**: Um objeto que define as configurações do e-mail que será enviado. Ele contém:
  - **from**: O e-mail remetente, que foi configurado como o e-mail armazenado na variável de ambiente (process.env.EMAIL_USER).
  - **to**: O destinatário do e-mail, que é passado como parâmetro para a função sendEmail.
  - **subject**: O assunto do e-mail, também passado como parâmetro.
  - **text**: O corpo do e-mail em formato de texto simples.

```
  transporter.sendMail(mailOptions, (error, info) => {
    if (error) {
      return console.log('Erro ao enviar e-mail:', error);
    }
    console.log('E-mail enviado:', info.response);
  });
};
```

- **transporter.sendMail(mailOptions, callback)**: Usa o transporter criado anteriormente para enviar o e-mail. Ele recebe:

- **mailOptions**: O objeto com as informações do e-mail (remetente, destinatário, assunto, corpo).
- **callback**: Uma função de callback que recebe dois argumentos:
  - **error**: Caso ocorra um erro ao tentar enviar o e-mail, o callback é chamado com o erro e é registrado no console.
  - **info**: Se o e-mail for enviado com sucesso, o callback registra no console a confirmação de que o e-mail foi enviado, mostrando a resposta do servidor de e-mail.

4. Exportação da Função sendEmail:

module.exports = { sendEmail };

- **module.exports**: Exporta a função sendEmail para que ela possa ser usada em outros arquivos no seu projeto.

Resumo do Funcionamento:

1. **Criação do transporter**: Um canal é configurado com o serviço de e-mail (Gmail, neste caso), usando as credenciais de e-mail e senha armazenadas nas variáveis de ambiente.
2. **Definição da função sendEmail**: Recebe o destinatário, assunto e corpo do e-mail e define as opções de envio.
3. **Envio do e-mail**: A função transporter.sendMail() é chamada para enviar o e-mail e lidar com os erros ou confirmações de envio.
4. **Uso seguro de credenciais**: O uso de variáveis de ambiente (process.env.EMAIL_USER e process.env.EMAIL_PASS) garante que as credenciais não estejam diretamente no código, tornando-o mais seguro.

Aqui no **authController.js** vamos implementar um sistema de **recuperação e redefinição de senha** em uma aplicação. Ele faz uso de **crypto** para gerar tokens, **bcrypt** para criptografar senhas, e o serviço de e-mail para enviar o link de redefinição de senha.

1. Importação de Módulos:

const crypto = require('crypto'); // Módulo para gerar tokens aleatórios de recuperação

```
const db = require('../config/db');  // Configuração da conexão com o banco de dados

const bcrypt = require('bcrypt');  // Para criptografar a nova senha

const sendEmail = require('../services/emailService').sendEmail;  // Função para enviar e-mails
```

- **crypto**: O módulo nativo do Node.js que gera valores aleatórios e seguros, como tokens de recuperação de senha.
- **db**: Conexão com o banco de dados.
- **bcrypt**: Para criptografar a senha antes de armazená-la no banco de dados.
- **sendEmail**: Função importada para enviar o e-mail ao usuário com o link para redefinição de senha.

2. Função requestPasswordReset:

Essa função é chamada quando o usuário solicita a recuperação de senha.

```
const requestPasswordReset = async (req, res) => {
 const { email } = req.body;
 try {
  const [user] = await db.promise().query('SELECT * FROM users WHERE email = ?', [email]);
  if (user.length === 0) {
   return res.status(404).send('Usuário não encontrado');
  }

  const token = crypto.randomBytes(20).toString('hex'); // Gera um token aleatório de 20 bytes
  const expireDate = new Date(Date.now() + 3600000); // Define o prazo de expiração (1 hora)

  await db.promise().query(
   'UPDATE users SET reset_password_token = ?, reset_password_expires = ? WHERE email = ?',
   [token, expireDate, email]
  );
```

```
    const resetLink = `http://localhost:3000/reset-password/${token}`; // Link para redefinição de senha

    sendEmail(email, 'Recuperação de Senha', `Por favor, clique no link para redefinir sua senha: ${resetLink}`);


    res.send('E-mail de recuperação de senha enviado');

  } catch (err) {

    console.error('Erro ao solicitar redefinição de senha:', err);

    res.status(500).send('Erro ao solicitar redefinição de senha');

  }

};
```

Explicação:

1. **const { email } = req.body;**: Extrai o e-mail fornecido no corpo da requisição.
2. **Verificação se o usuário existe no banco de dados**:
    - O banco de dados é consultado para verificar se existe um usuário registrado com o e-mail fornecido.
    - Se não for encontrado, retorna 404 - Usuário não encontrado.
3. **Geração do Token de Recuperação**:
    - **crypto.randomBytes(20).toString('hex')**: Gera um token aleatório de 20 bytes e o converte para uma string hexadecimal.
    - **expireDate**: Define a data de expiração para 1 hora a partir do momento atual.
4. **Armazenamento do Token e Data de Expiração no Banco de Dados**:
    - O token e a data de expiração são salvos no banco de dados associados ao usuário.
5. **Envio do E-mail com o Link de Recuperação**:
    - Um e-mail é enviado para o usuário contendo um link com o token gerado. Esse link será usado para redefinir a senha.
6. **Tratamento de Erros**:
    - Se ocorrer algum erro durante o processo, ele é capturado e uma resposta de erro 500 - Erro ao solicitar redefinição de senha é enviada.

3. Função resetPassword:

Essa função é chamada quando o usuário acessa o link de redefinição de senha e fornece uma nova senha.

```
const resetPassword = async (req, res) => {
  const { token, newPassword } = req.body;
  try {
    const [user] = await db.promise().query(
      'SELECT * FROM users WHERE reset_password_token = ? AND reset_password_expires > NOW()',
      [token]
    );

    if (user.length === 0) {
      return res.status(400).send('Token inválido ou expirado');
    }

    const hashedPassword = await bcrypt.hash(newPassword, 10); // Criptografa a nova senha
    await db.promise().query(
      'UPDATE users SET password = ?, reset_password_token = NULL, reset_password_expires = NULL WHERE id = ?',
      [hashedPassword, user[0].id]
    );

    res.send('Senha redefinida com sucesso');
  } catch (err) {
    console.error('Erro ao redefinir senha:', err);
    res.status(500).send('Erro ao redefinir senha');
  }
};
```

Explicação:

1. **const { token, newPassword } = req.body;**: Extrai o token e a nova senha fornecidos no corpo da requisição.
2. **Verificação do Token**:
    - O banco de dados é consultado para encontrar o usuário cujo token de redefinição corresponde ao token fornecido e que ainda está dentro do prazo de expiração (reset_password_expires > NOW()).
    - Se o token for inválido ou expirado, retorna 400 - Token inválido ou expirado.
3. **Criptografia da Nova Senha**:
    - **bcrypt.hash(newPassword, 10)**: A nova senha fornecida é criptografada antes de ser armazenada no banco de dados.
4. **Atualização do Banco de Dados**:
    - A nova senha criptografada substitui a antiga no banco de dados.
    - O token de redefinição e a data de expiração são removidos (definidos como NULL) para que o token não possa ser reutilizado.
5. **Resposta de Sucesso**:
    - Se o processo for bem-sucedido, a mensagem Senha redefinida com sucesso é enviada.
6. **Tratamento de Erros**:
    - Qualquer erro encontrado durante o processo gera uma resposta 500 - Erro ao redefinir senha.

4. Exportação das Funções:

module.exports = { requestPasswordReset, resetPassword };

- As funções requestPasswordReset e resetPassword são exportadas para serem usadas em outras partes do código (geralmente em rotas).

Resumo do Funcionamento:

1. **Solicitação de Redefinição de Senha** (requestPasswordReset):
    - O usuário fornece seu e-mail.
    - Se o e-mail for encontrado, um token de redefinição de senha é gerado, armazenado no banco de dados e enviado ao usuário por e-mail com um link para redefinir sua senha.

2. **Redefinição de Senha** (resetPassword):
    - O usuário acessa o link enviado para o e-mail e fornece uma nova senha.
    - O sistema verifica se o token é válido e não expirou.
    - Se for válido, a senha é criptografada e atualizada no banco de dados, e o token é invalidado.

    Esse fluxo é comum em sistemas que oferecem recuperação de senha segura, garantindo que o token expire e que a senha seja armazenada de forma criptografada.