Aula 8

Created	@January 21, 2024 4:58 PM
Data	@January 22, 2024
Status	Não iniciado
Tipo	Aula

Projeto - Loja Virtual (Login do Cliente)

Autenticação:

A autenticação é o processo de verificar a identidade de um usuário, sistema ou aplicativo. O objetivo é garantir que a pessoa ou entidade que está acessando o sistema seja quem ela afirma ser. Os métodos comuns de autenticação incluem:

1. **Nome de Usuário e Senha:**
    - A combinação mais comum, onde o usuário fornece um nome de usuário e uma senha.
2. **Token de Acesso:**
    - O usuário recebe um token (geralmente após o login) e o apresenta em solicitações subsequentes para provar a autenticação.
3. **Autenticação Multifatorial (MFA):**
    - Envolve o uso de mais de um método de autenticação (por exemplo, senha e código enviado por SMS).

4. **Biometria:**
    - Utiliza características físicas únicas do usuário, como impressão digital, reconhecimento facial ou íris.

Autorização:

A autorização é o processo de determinar se um usuário autenticado tem permissão para realizar uma determinada ação ou acessar um recurso específico. Depois que um usuário é autenticado, a autorização determina o que ele pode fazer no sistema. Métodos comuns de autorização incluem:

1. **Controle de Acesso Baseado em Funções (RBAC):**
    - Atribui funções aos usuários e concede permissões com base nessas funções.
2. **Controle de Acesso Baseado em Política (ABAC):**
    - Usa políticas flexíveis que levam em consideração várias variáveis (atributos) para tomar decisões de autorização.
3. **Token JWT (JSON Web Token):**
    - Inclui informações sobre as permissões do usuário no token, facilitando a autorização em sistemas distribuídos.
4. **Listas de Controle de Acesso (ACL):**
    - Associa diretamente usuários a recursos específicos e especifica as permissões.

Panorama Geral:

1. **Ciclo de Vida:**
    - A autenticação ocorre no início da interação, enquanto a autorização acontece conforme o usuário tenta acessar recursos ou realizar ações.
2. **Segurança de Transporte:**
    - Em ambientes online, é essencial proteger os dados durante a autenticação e autorização. O uso de HTTPS e outros protocolos seguros é crucial.
3. **OAuth e OpenID Connect:**

- São padrões populares para autenticação e autorização em aplicativos da web e APIs.

4. **Segurança Contextual:**
   - Em ambientes modernos, a segurança muitas vezes depende do contexto, considerando fatores como localização, dispositivo e comportamento do usuário.
5. **Logs e Auditoria:**
   - A implementação de registros e auditorias é essencial para monitorar atividades e identificar possíveis ameaças ou violações de segurança.
6. **Proteção contra Ataques:**
   - Medidas devem ser implementadas para proteger contra ataques comuns, como força bruta, falsificação de identidade, entre outros.

Middlewares

Middleware é uma camada intermediária entre o sistema operacional e o aplicativo, ou entre diferentes componentes de um aplicativo. Ele desempenha um papel fundamental na manipulação de solicitações HTTP, execução de lógica de negócios, gerenciamento de estado, autenticação, autorização e outros aspectos relacionados ao processamento de requisições. Vamos explorar alguns aspectos importantes do uso de middleware em uma aplicação:

1. Middleware em Pilha:

- Middleware é frequentemente organizado em uma pilha, onde cada camada executa uma função específica. A ordem em que os middleware são colocados na pilha é crucial, pois determina a sequência de execução.

2. Funções Comuns do Middleware:

- **Log e Auditoria:** Middleware pode ser usado para registrar detalhes sobre as solicitações e respostas, facilitando a auditoria e a resolução de problemas.

- **Autenticação e Autorização:** Middleware pode verificar a autenticação dos usuários e garantir que tenham as permissões adequadas para acessar recursos específicos.
- **Processamento de Corpo da Solicitação:** Middleware pode analisar e processar o corpo das solicitações, realizando validações ou transformações necessárias.
- **Manipulação de Erros:** Middleware pode capturar e lidar com erros, fornecendo respostas apropriadas ao cliente ou registrando eventos para análise posterior.

3. Express Middleware:

- No contexto do Node.js e Express, middleware é uma parte essencial. Express tem um sistema de middleware robusto que permite adicionar funções intermediárias apropriadas para diferentes etapas do processamento da requisição.

```
javascriptCopy code
const express = require('express');
const app = express();

// Middleware para log de solicitações
app.use((req, res, next) => {
  console.log(`Solicitação recebida: ${req.method} ${req.url}
`);
  next(); // Chama o próximo middleware na pilha
});

// Middleware para autenticação
app.use((req, res, next) => {
  if (req.headers.authorization === 'token-secreto') {
    next(); // Usuário autenticado
  } else {
    res.status(401).send('Não autorizado');
  }
});

// Rotas e manipuladores de rota vêm após os middlewares
```

```
app.get('/', (req, res) => {
  res.send('Bem-vindo à minha aplicação!');
});

// ...

app.listen(3000, () => {
  console.log('Servidor iniciado na porta 3000');
});
```

4. Terceirização de Middleware:

- Pacotes de middleware de terceiros são amplamente utilizados para adicionar funcionalidades específicas. Por exemplo, middleware de compressão para compactar respostas, middleware de segurança para proteger contra ataques, etc.

```
javascriptCopy code
const compression = require('compression');
const helmet = require('helmet');

app.use(compression()); // Middleware para compactar resposta
s
app.use(helmet()); // Middleware para melhorar a segurança
```

5. Contexto de Middleware:

- Middleware têm acesso ao objeto de requisição (`req`), objeto de resposta (`res`), e uma função `next` para encaminhar a solicitação para o próximo middleware na pilha.

6. Uso em Microsserviços:

- Em arquiteturas de microsserviços, o uso de middleware é comum para implementar lógica compartilhada, autenticação entre serviços e manipulação de

erros.

7. Desafios Potenciais:

- Uma ordem incorreta de middleware pode levar a resultados inesperados.
- Middleware excessivo pode prejudicar o desempenho.

8. Testabilidade:

- O uso adequado de middleware facilita a modularização e teste de diferentes partes da aplicação de forma isolada.

Instalando e configurando o express-session

Para instalar e configurar o `express-session`, você precisará do Node.js e do npm instalados em seu ambiente. Aqui estão os passos básicos para instalar e configurar o `express-session` em uma aplicação Express:

1. Criar um Projeto Node.js:

Se você ainda não tiver um projeto Node.js, crie um novo diretório e execute o seguinte comando para iniciar um projeto Node.js:

```
bashCopy code
npm init -y
```

2. Instalar o Express e o express-session:

Instale o Express e o express-session usando o npm:

```
bashCopy code
npm install express express-session
```

3. Configurar e Usar o express-session no seu Aplicativo:

Crie um arquivo JavaScript (por exemplo, `app.js`) para o seu aplicativo e configure o `express-session`:

```
javascriptCopy code
const express = require('express');
const session = require('express-session');

const app = express();

// Configurar o express-session
app.use(session({
  secret: 'seuSegredoAqui', // Segredo para assinar a sessão
(mantenha-o seguro)
  resave: false,
  saveUninitialized: true,
  cookie: { secure: false } // Defina como true se estiver us
ando HTTPS
}));

// Defina uma rota para demonstrar o uso do express-session
app.get('/', (req, res) => {
  // Acesse ou configure valores na sessão
  if (req.session.views) {
    req.session.views++;
  } else {
    req.session.views = 1;
  }

  res.send(`Número de visualizações: ${req.session.views}`);
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Servidor iniciado na porta ${PORT}`);
```

```
});
```

4. Executar o Aplicativo:

Execute o aplicativo usando o seguinte comando:

```
bashCopy code
node app.js
```

Se tudo estiver configurado corretamente, seu aplicativo estará disponível em `http://localhost:3000/` (ou outra porta especificada) e você verá o número de visualizações incrementando cada vez que acessa a rota raiz.

Notas Importantes:

- Certifique-se de manter o segredo (`secret`) seguro. É usado para assinar as sessões e deve ser mantido em segredo para garantir a segurança.
- O exemplo acima usa a memória para armazenar as sessões, o que é útil apenas para fins de demonstração. Para um ambiente de produção, você pode querer usar armazenamento externo, como o express-session-store.
- Configure o atributo `cookie.secure` como `true` se estiver usando HTTPS para garantir que os cookies da sessão sejam transmitidos apenas por meio de conexões seguras.

Certifique-se de adaptar a configuração do `express-session` conforme necessário para atender aos requisitos específicos do seu aplicativo.

Instalando e configurando o cookie-parser

O `cookie-parser` é usado para analisar cookies em solicitações HTTP. Aqui estão os passos para instalar e configurar o `cookie-parser` em uma aplicação Express:

1. Instalar o cookie-parser:

Certifique-se de estar no diretório do seu projeto e execute o seguinte comando para instalar o `cookie-parser`:

```
bashCopy code
npm install cookie-parser
```

2. Configurar e Usar o cookie-parser no seu Aplicativo:

Crie ou edite o arquivo JavaScript do seu aplicativo (por exemplo, `app.js`) e configure o `cookie-parser`:

```
javascriptCopy code
const express = require('express');
const cookieParser = require('cookie-parser');

const app = express();

// Usar o cookie-parser
app.use(cookieParser());

// Definir uma rota para demonstrar o uso do cookie-parser
app.get('/', (req, res) => {
  // Ler cookies da requisição
  const visitCount = req.cookies.visitCount || 0;

  // Incrementar o contador de visitas e definir o cookie
  res.cookie('visitCount', visitCount + 1, { maxAge: 900000,
httpOnly: true });

  res.send(`Número de visitas: ${visitCount}`);
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Servidor iniciado na porta ${PORT}`);
```

```
});
```

Neste exemplo, o middleware `cookie-parser` é utilizado para analisar os cookies da requisição. O valor do cookie `visitCount` é lido, incrementado e então definido novamente na resposta. Este é apenas um exemplo básico; você pode personalizar conforme necessário para atender aos requisitos do seu aplicativo.

Notas Importantes:

- Certifique-se de ajustar as opções do cookie conforme necessário (por exemplo, `maxAge`, `httpOnly`).
- O exemplo acima usa o middleware `cookie-parser` antes de definir as rotas, garantindo que seja aplicado a todas as solicitações.

Certifique-se de adaptar a configuração do `cookie-parser` conforme necessário para atender aos requisitos específicos do seu aplicativo.

Prático:

- Instalar e configurar o módulo express-session;
  - Ref: https://expressjs.com/en/resources/middleware/session.html
- Instalar e configurar o módulo cookie-parser;
- Adicionar uma rota /login para geração do cookie de sessão
- Adicionar uma rota /logout para exclusão do cookie de sessão
- Implementar um middleware de autorização do cookie de sessão
- Adicionar o middleware as rotas que devem ser protegidas