



Prof. David Fernandes de Oliveira
Instituto de Computação
UFAM

Desenvolvendo CRUDs

- Uma parte importante do desenvolvimento de uma aplicação é a criação do **CRUD** a alguns modelos
 - O CRUD de um modelo é um conjunto de páginas responsáveis por quatro operações sobre esse esse modelo:



Desenvolvendo CRUDs

- Para exemplificar a criação de novos CRUDs, vamos desenvolver um para o modelo Curso
- O primeiro passo é criar um controlador vazio para o CRUD de Curso dentro do diretório **/src/controllers**

Operações
do CRUD

```
// Arquivo src/controllers/departamento.js  
import { Request, Response } from 'express';  
import { Departamentos } from '../models/Departamentos';  
  
async function index (req: Request, res: Response) {};  
async function read (req: Request, res: Response) {};  
async function create (req: Request, res: Response) {};  
async function update (req: Request, res: Response) {};  
async function remove (req: Request, res: Response) {};  
  
export default { index, read, create, update, remove }
```

Definindo as Rotas do CRUD

- O segundo passo é definir as rotas para cada operação do CRUD:

```
import { Router } from 'express';
import deptController from '../controllers/departamento';

const router = Router();

// Departamento controller
router.get('/dept', deptController.index);
router.get('/dept/create', deptController.create);
router.post('/dept/create', deptController.create);
router.get('/dept/update/:id', deptController.update);
router.post('/dept/update/:id', deptController.update);
router.get('/dept/:id', deptController.read);
router.post('/dept/:id', deptController.remove);

export default router;
```

Definindo as Rotas do CRUD

- O segundo passo é definir as rotas para cada operação do CRUD:

```
import { Router } from 'express';
import deptController from '../controllers/departamento';

const router = Router();

// Departamento controller
router.get('/dept/:id', deptController.index);
router.post('/dept/:id', deptController.create);
router.get('/dept/update/:id', deptController.update);
router.post('/dept/update/:id', deptController.update);
router.get('/dept/:id', deptController.read);
router.post('/dept/:id', deptController.remove);

export default router;
```

Embora não faça parte do CRUD, o objetivo da rota **/curso** é listar os cursos existentes

Definindo as Rotas do CRUD

- O segundo passo é definir as rotas para cada operação do CRUD:

```
import { Router } from 'express';
import deptController from '../controllers/departamento';

const router = Router();

// Departamento controller

router.get('/dept/:id', deptController.read);
router.post('/dept/:id', deptController.remove);

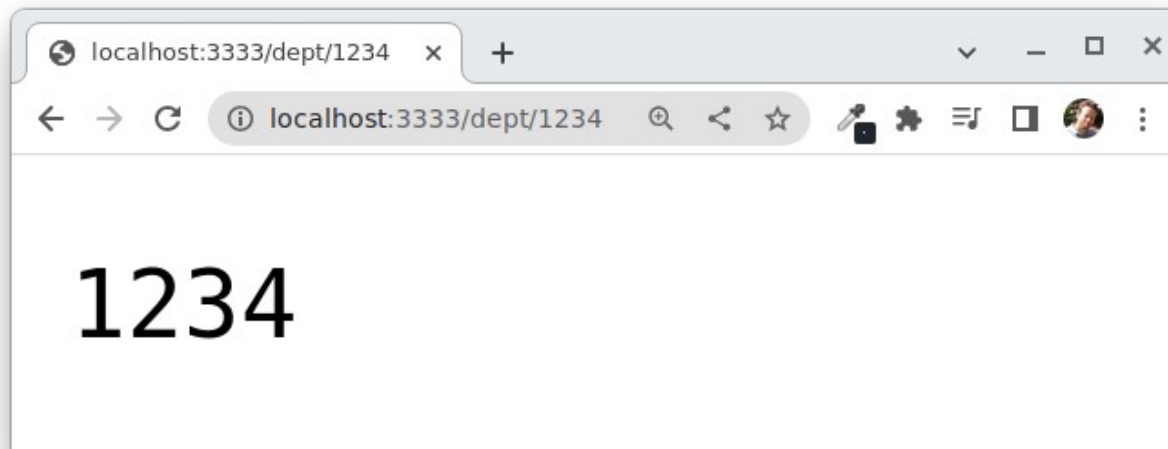
export default router;
```

Note que as rotas para **read**, **update** e **remove** terminam com a string **:id**, que representa um parâmetro utilizado para informar que curso se deseja **ler**, **atualizar** ou **apagar**

Definindo as Rotas do CRUD

- Por exemplo, na url **http://localhost:3000/dept/1234**, o valor do parâmetro **id** é 1234
- Para ler o valor de **id** dentro da action, podemos usar o atributo **param** de **req** (objeto da requisição do usuário):

```
const read = async (req: Request, res: Response) {  
  const { id } = req.params;  
  res.end(id);  
},
```



Requisições POST

- Quando o usuário preenche e submete um formulário POST, os dados informados pelo usuário são enviados para o servidor

```
<form action="/dept/create" method="post">  
  <input type="text" name="nome">  
  <input type="text" name="sigla">  
  <input type="submit" value="Enviar">  
</form>
```

- Após a submissão, os dados são enviados através do corpo da requisição HTTP (request body)
- Esses dados são enviados no formato **application/x-www-form-urlencoded**, que também é usado em requisições GET:

```
sigla=VE&nome=Vendas
```


Requisições POST

- O Express possui um middleware nativo chamado **urlencoded**, que pode ser usado para extrair os dados de **request body**
- Para usá-lo, basta inserir a linha abaixo no arquivo **index.ts**, em alguma lugar antes da chamada ao middleware router:

```
// Arquivo index.ts  
app.use(express.urlencoded({extended: false}));  
...  
app.use(router);
```

Requisições POST

- Após isso, o **urlencoded** irá extrair os dados do **request body** de todas as requisições POST e copiá-los no objeto **req.body**

```
const create = async (req: Request, res: Response) => {  
  if (req.route.methods.get) {  
    res.render('departamento/create');  
  } else {  
    const dept: DepartamentoDto = {  
      id: uuidv1(),  
      ...req.body,  
      createdAt: new Date(),  
      updatedAt: new Date(),  
    };  
    await Departamentos.create(dept);  
    res.redirect('/dept');  
  }  
};
```

Regras de Validação

- Os **validadores** são decorators usados nos modelos para garantir a consistência dos dados durante a inserção e atualização dos registros

Validador	Descrição	Exemplo
@IsInt	Verifica se um dado valor é um inteiro	@IsInt
@Max	Verifica se um dado número é menor que um dado valor	@max(10000)
@Min	Verifica se um dado número é maior que um dado valor	@Min(0)
@IsEmail	Verifica se um dado valor é um email válido ou não	@IsEmail

Regras de Validação

- Os **validadores** são decorators usados nos modelos para garantir a consistência dos dados durante a inserção e atualização dos registros

Outros exemplos de validadores: `@IsUrl`, `@IsIP`, `@IsAlpha`, `@IsAlphanumeric`, `@IsNumeric`, `@IsFloat`, `@IsDecimal`, `@IsLowercase`, `@IsUppercase`, `@NotNull`, `@IsNull`, `@NotEmpty`, `@Equals`, `@Contains`, `@NotIn`, `@NotContains`, `@Length`, `@IsDate`, `@IsAfter`, `@IsBefore`, `@IsArray`, `@IsCreditCard`, dentre outros.

A documentação completa dos validadores pode ser encontrada na **documentação oficial do sequelize**

@IsEmail

Verifica se um dado valor é um email válido ou não

@IsEmail

Regras de Validação

- Os **validadores** são decorators usados nos modelos para garantir a consistência dos dados durante a inserção e atualização dos registros.

Outros exemplos de validadores: **@IsAlpha**, **@IsAlphanumeric**, **@IsLowercase**, **@IsUnique**, **@Equals**, **@Contains**, **@IsAfter**, **@IsBefore**

A documentação completa dos validadores pode ser encontrada na **documentação oficial do sequelize**

```
@Length({
  max: 3,
  min: 50,
})
@AllowNull(false)
@Unique
@Column({
  type: DataType.STRING,
})
name!: string;
```

@IsEmail

Verifica se um dado valor é um email válido ou não

@IsEmail

Regras de Validação

- O módulo de validação usado pelo Sequelize possui mensagens de erro padronizadas
- No entanto, é possível definir mensagens personalizadas para cada um dos validadores apresentados

```
@isInt({ msg: "Precisa ser um número inteiro" })
```

```
@Length({  
  max: 3,  
  min: 50,  
  msg: 'O nome precisa conter entre 3 e 50 caracteres',  
})
```

Regras de Validação

- Para exemplificar o uso dos validadores, considere que o nome dos departamentos precisa conter entre 3 e 40 caracteres

```
@Length({  
  max: 3,  
  min: 50,  
  msg: 'O nome precisa conter entre 3 e 50 caracteres',  
})  
@AllowNull(false)  
@Unique  
@Column({  
  type: DataType.STRING,  
})  
name!: string;
```

Regras de Validação

- Para exemplificar o uso dos validadores considere que o nome dos

@Le
m
m
m
})
@Al
@Un
@Co
t
})
nam

```
const create = async (req: Request, res: Response) => {  
  if (req.route.methods.get) {  
    res.render('departamento/create');  
  } else {  
    const dept: DepartamentoDto = {  
      id: uuidv1(),  
      ...req.body,  
      createdAt: new Date(),  
      updatedAt: new Date(),  
    };  
    try {  
      await Departamentos.create(dept);  
      res.redirect('/dept');  
    } catch (e: any) {  
      console.log(e.errors);  
    }  
  }  
};
```

Caso o nome do curso tenha um número inválido de caracteres, o **catch** será acionado

Desafio de Validação

Minha Loja

localhost:3333/dept/create

Minha Loja

Adicionar Departamento

Nome do Departamento

Co

Sigla do Departamento

CO

Submeter

Recurso de Validação

npm start ~/d/expTS

localhost:3333/dept/create

```
ValidationErrorItem {
  message: '0 nome precisa conter entre 3 e 50 caracteres',
  type: 'Validation error',
  path: 'name',
  value: 'Co',
  origin: 'FUNCTION',
  instance: Departamento {
    dataValues: [Object],
    _previousDataValues: [Object],
    unico: 1,
    _changed: [Set],
    _options: [Object],
    isNewRecord: true
  },
  validatorKey: 'len',
  validatorName: 'len',
  validatorArgs: [ 50, 3 ],
  original: Error: 0 nome precisa conter entre 3 e 50 caracteres
```

Submeter

Regras de Validação

- Para mostrar os erros na view **dept/create**, teremos que usar a função **render** passando o vetor de erros ocorridos

```
const create = async (req: Request, res: Response) => {  
  ...  
  try {  
    await Departamentos.create(dept);  
    res.redirect('/dept');  
  } catch (e: any) {  
    console.log(e.errors);  
    res.render('dept/create', { dept, errors: e.errors });  
  }  
  ...  
};
```

Regras de Validação

- Podemos usar **helpers handlebars** para mostrar os erros para os usuários

```
<div class="form-group">
  <label for="name">Nome do Departamento</label>
  <input type="text" name="name" value="{{dept.name}}">
  <div class="invalid-feedback">{{showError errors 'name'}}</div>
</div>
```

```
export function showError(errors: any[], field: string) {
  let mensagem = '';
  if (errors) {
    errors.forEach((e) => {
      if (e.path === field) {
        mensagem += e.message;
      }
    });
  }
  return mensagem;
}
```

Deposito de Validação

Minha Loja

+

localhost:3333/dept/create

Minha Loja

Adicionar Departamento

Nome do Departamento

Co

O nome precisa conter entre 3 e 50 caracteres.

Sigla do Departamento

CO

Submeter

Desafio de Validação

Minha Loja

localhost:3333/dept/create

Adicionar Departamento

Nome do Departamento

O nome precisa conter entre 3 e 50 caracteres.

Sigla do Departamento

Submeter

Exercício: Crie os CRUDs para os modelos **Departamentos** e **Funcionários**. Faça validação de todos os dados dos Formulários.

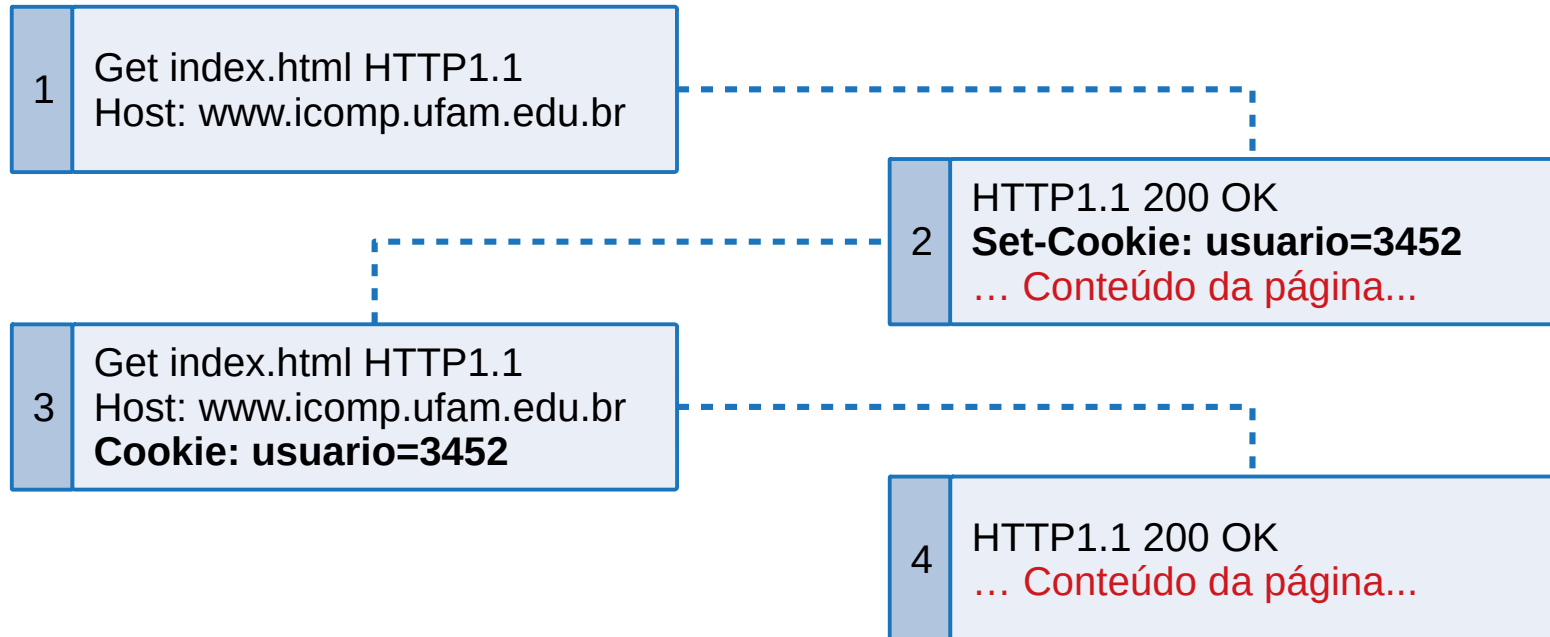
Cookies



- **Cookies** são variáveis enviadas pelo servidor Web para o browser através do **protocolo HTTP**
 - Ficam armazenados no lado cliente
 - São enviados para o servidor em futuros acessos do browser

Requisições do Browser

Requisições de **icomp.ufam.edu.br**



Cookies



- Para habilitar o uso de cookies em sua aplicação, é necessário instalar o middleware **cookie-parser**

```
$ npm install cookie-parser  
$ npm install -D @types/cookie-parser
```

- Para usar o middleware, precisamos dar importação ao módulo e adicioná-lo em nossa aplicação com o método **use**

```
// Arquivo src/server.ts  
import cookieParser from 'cookie-parser';  
  
// Método middleware():  
this.server.use(cookieParser())
```


Cookies



- A partir deste momento, podemos i) criar novos cookies e ii) identificar os cookies enviados pelo browser para o servidor
 - O array **req.cookies** armazena os cookies enviados pelo browser
 - O método **res.cookie** é usado para enviar um pedido de criação de um novo cookie no browser

```
// Arquivo src/router/router.ts  
router.get('/create-cookie', mainController.createCookie);
```

```
// Arquivo src/controllers/main.ts  
const createCookie = function (req: Request, res: Response) {  
  if (!('nomeCookie' in req.cookies)) {  
    res.cookie('nomeCookie', 'valorCookie');  
    res.send('Você NUNCA passou por aqui!');  
  } else {  
    res.send('Você JÁ passou por aqui');  
  }  
};
```



Você NUNCA
passou por
aqui

Requisição
para criação
do novo
cookie

Ao acessar a página pela
primeira vez, aparece a
mensagem: **Você nunca
passou por aqui.**

localhost:3333/create-cookie

Elements Network

Preserve log

No throttling

Filter

All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest

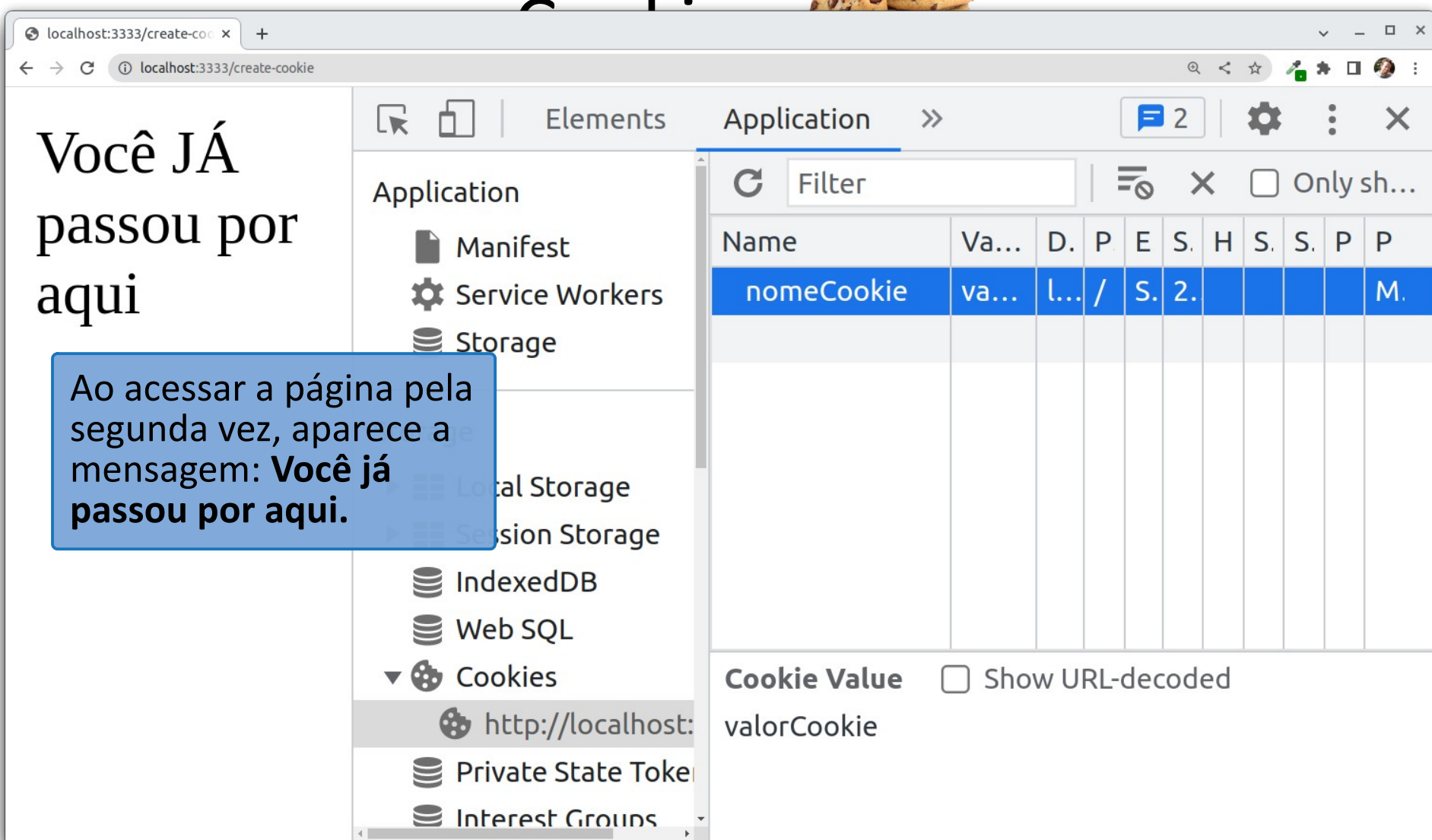
20 ms 40 100 ms

Set-Cookie: nomeCookie=valorCookie;

Path=

Express

Select a cookie to preview its value



Cookies



- Também podemos criar cookies com data de expiração

```
// Expira 360000 ms (6 minutos) após ser criado  
res.cookie(name, 'value', { maxAge: 360000 } );
```

- Usamos a função **clearCookie** para apagar um cookie já criado

```
// Arquivo src/router/router.ts  
router.get('/clear-cookie', mainController.clearCookie);
```

```
// Arquivo src/controllers/main.ts  
const clearCookie = function (req: Request, res: Response) {  
  res.clearCookie('nome');  
  res.send('cookie apagado');  
});
```

Cookies



- Também podemos criar cookies com data de expiração

```
// Expira 360000 ms (6 minutos) após ser criado  
res.cookie(name, 'value', { maxAge: 360000 } );
```

- Usamos a função **clearCookie** para apagar um cookie já criado

```
// Apagar cookie criado sem data de expiração  
router.get('/', (req, res) => {  
  // Se o cookie for criado sem data de expiração, ele será  
  // apagado após o fechamento da janela do browser  
  res.clearCookie('nome');  
  res.send('cookie apagado');  
});
```

```
// Arquivo src/controllers/main.ts  
const clearCookie = function (req: Request, res: Response) {  
  res.clearCookie('nome');  
  res.send('cookie apagado');  
});
```

Cookies

A screenshot of a web browser window. The address bar shows 'localhost:3333/login'. The page has a dark blue header with 'Minha Loja' and navigation links 'Sobre', 'UI', 'Departamentos', 'Login', and 'Cadastro'. The main content area is dark grey with the title 'Acesso de Usuário'. Below the title are two white input fields labeled 'Username' and 'Senha'. A blue 'Entrar' button is at the bottom left of the form.

Exercício: Usando **cookies**, crie um mecanismo de login e logout para o site. Considere que o site possua apenas um usuário, com username **user** e senha **12345**. Quando o usuário loga, a opção de **Login** do menu superior é substituída por uma opção de **Logout**. O usuário só poderá acessar o CRUD de departamento caso esteja Logado.

Cookies



- Para fazer o exercício anterior, podemos criar novas rotas de login e logout associadas ao controlador main

```
// Arquivo src/router/router.ts  
router.get('/login', mainController.login);  
router.post('/login', mainController.login);  
router.get('/logout', mainController.logout);
```

- A action Login deverá imprimir o formulário de login na rota **GET /login** e deverá processar o login na rota **POST /login**
 - Na rota POST /login, um cookie booleano **logado** deverá ser criado caso as credenciais do usuário estejam corretas
- A action Logout deverá usar a função **clearCookie** para apagar o cookie **logado** e redirecionar o usuário para a tela de login

Cookies



- Caso as credenciais do usuário estejam corretas, podemos mudar o menu superior conforme a imagem abaixo

Minha Loja Sobre UI Departamentos

Logout

- Para isso, o **handlebars** precisa saber se o usuário está logado ou não, o que pode ser feito através do objeto **res.locals**
- Quando usamos um middleware para adicionar propriedades no objeto **res.locals**, essas propriedades passam a ser visíveis dentro das views e layouts do handlebars

Cookies



- Por exemplo, ao adicionarmos o middleware **setLocals** abaixo em nossa aplicação, a propriedade **logado** passa a ser visível dentro do layout do handlebars

```
// Arquivo src/middlewares/setLocals.ts  
export const setLocals = (req, res, next) => {  
  res.locals.logado = req.cookies['logado'];  
  next();  
};
```

```
<!-- Arquivo src/views/layouts/main.handlebars -->  
{{#if logado}}  
  <li class='nav-item'>  
    <a class='nav-link' href='/logout'>Logout</a>  
  </li>  
{{/if}}
```

Cookies



- Para impedir que usuários não logados acessem as páginas do CRUD de departamento, podemos usar o middleware abaixo

```
// Arquivo src/middlewares/checkAuth.ts  
const checkAuth = (req, res, next) => {  
  const logado = req.cookies['logado'];  
  if (!logado) res.redirect('/login');  
  else next();  
};
```

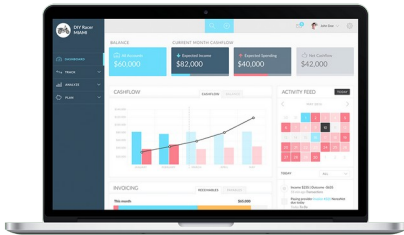
- Adicionando o middleware **checkAuth** nas rodas do CRUD de departamento, impedimos que os usuários acessem tais rotas

```
// Arquivo src/router/router.ts  
router.get('/dept', checkAuth, deptController.index);  
router.get('/dept/create', checkAuth, deptController.create);  
router.post('/dept/create', checkAuth, deptController.create);
```

Cross-Site Request Forgery

- Um ataque clássico a aplicações Web é o **Cross Site Request Forgery**, também conhecido pelas siglas **CSRF** ou **XSRF**

www.bancoficticio.com



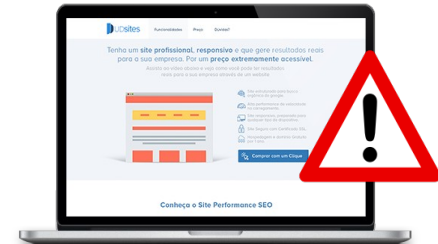
João faz o login em sua conta do e-banking



João recebe em um e-mail de spam e clica em um link que redireciona para o site Attacker



www.attacker.com

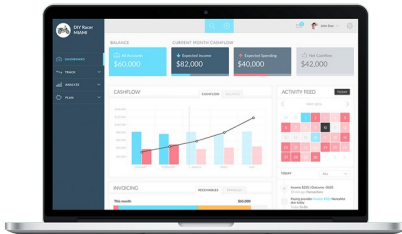


Attacker aproveita que João está logado no banco e faz um solicitação de transferência

Cross-Site Request Forgery

- Um ataque clássico a aplicações Web é o **Cross Site Request Forgery**, também conhecido pelas siglas **CSRF** ou **XSRF**

www.bancoficticio.com



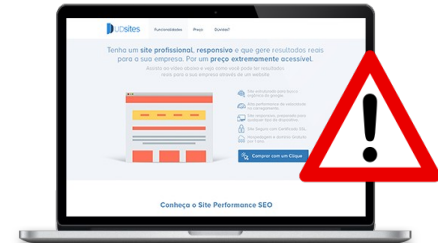
João faz o login em sua conta do e-banking



João recebe em um e-mail de spam e clica em um link que redireciona para o site Attacker



www.attacker.com



Attacker aproveita que João está logado no banco e faz um solicitação de transferência

```
<form action="https://www.bancoficticio.com" method="POST">
  <input name="destinatario" value="attacker">
  <input name="quantia" value="10.000,00">
</form>
<script>
  document.forms[0].submit()
</script>
```

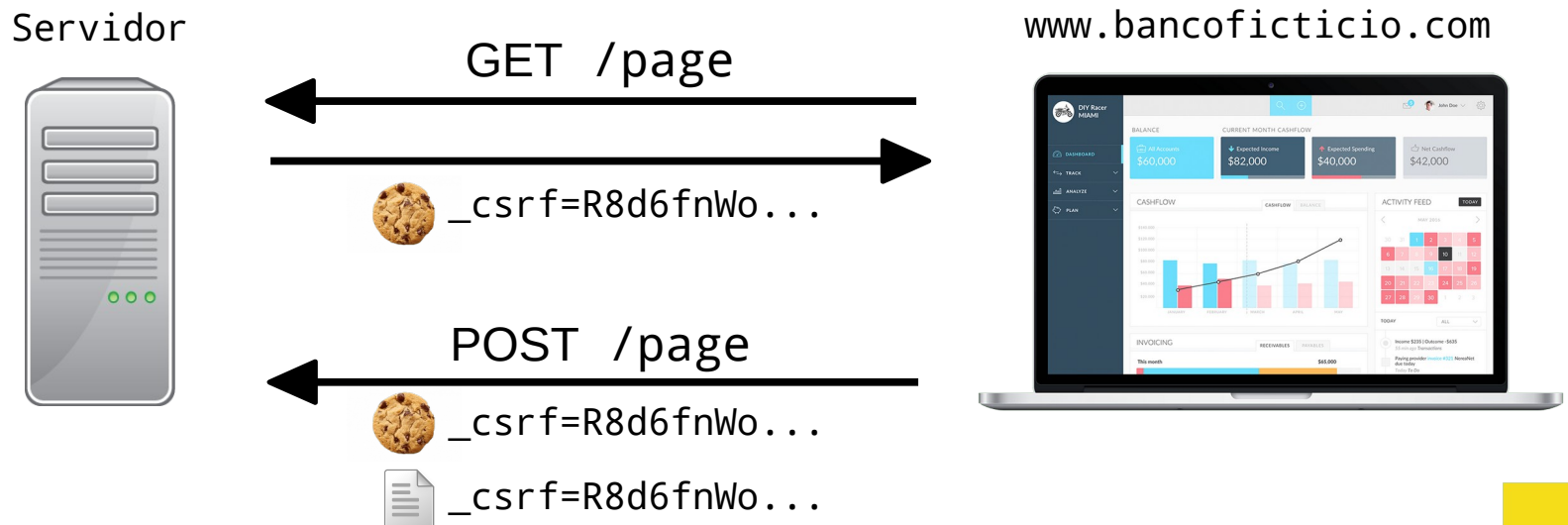
HACKED

express

JS

Cross-Site Request Forgery

- Uma forma de proteger os usuários de sua aplicação Web desse tipo de ataque é através dos **tokens CSRF**
- Um token CSRF é um conjunto de 24 caracteres gerado aleatoriamente – Ex: Jq6nXu-4oqaEDluG7XOCCWO
- Um novo token CSRF é enviado como cookie para o browser sempre que o usuário acessar uma página do servidor Web

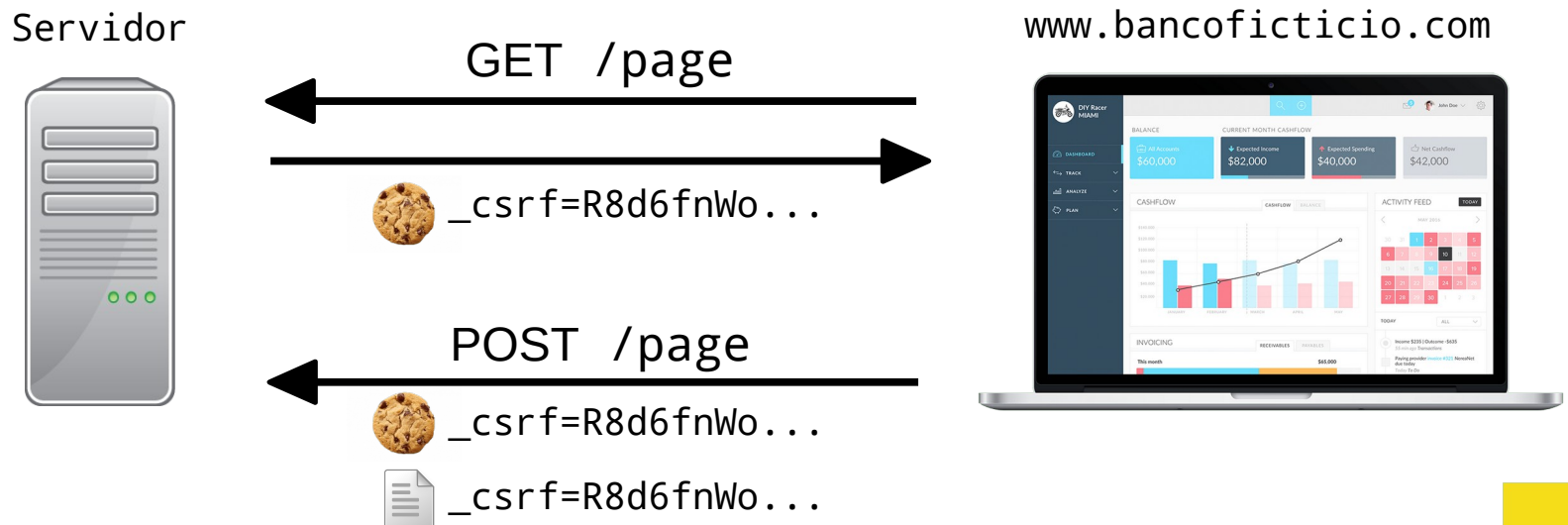


Cross-Site Request Forgery

- Além do cookie, o token CSRF também é colocando como um **input hidden** de todo formulário disponível no servidor

```
<input type="hidden" name="_csrf" value="{{csrf}}">
```

- Portanto, quando o usuário preenche o formulário, o CSRF é enviado para o server como cookie e como input do formulário



Cross-Site Request Forgery

- Após a submissão de um formulário, o servidor irá comparar o CSRF vindo do formulário com o CSRF vindo do cookie
- Os dados do formulário só serão processados caso os dois CSRFs sejam exatamente iguais

Servidor



GET /page



_csrf=R8d6fnWo...

POST /page

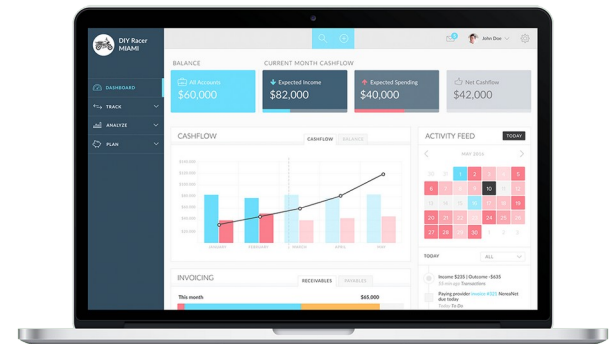


_csrf=R8d6fnWo...



_csrf=R8d6fnWo...

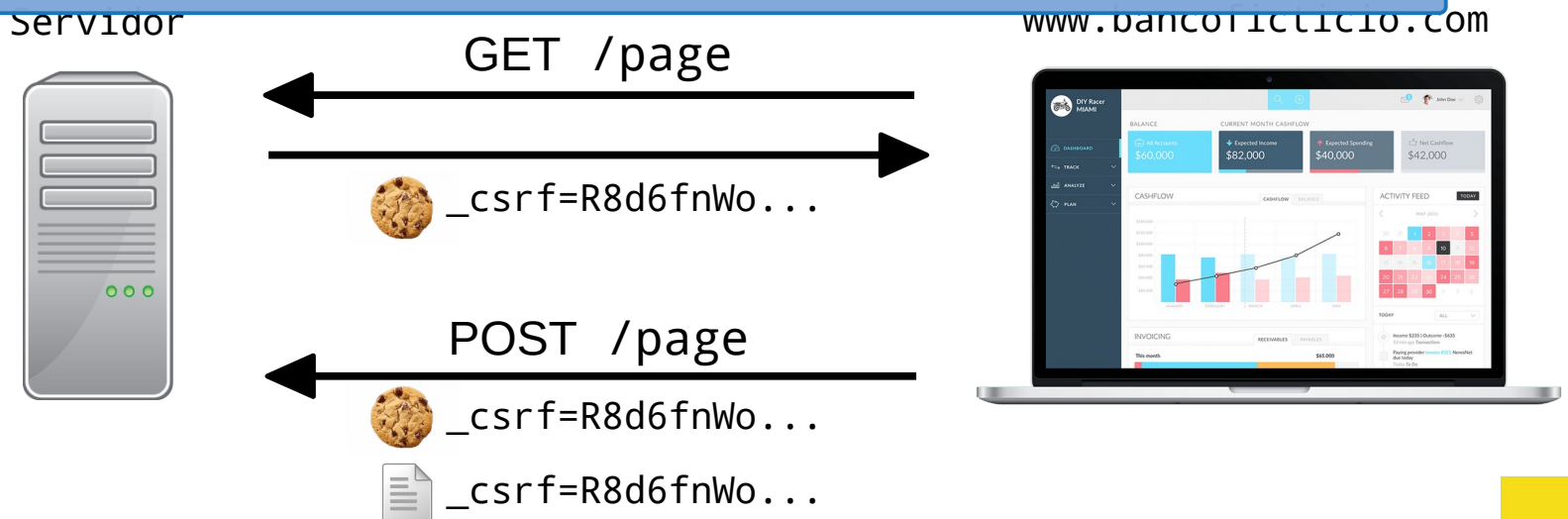
www.bancoficticio.com



Cross-Site Request Forgery

- Após a submissão de um formulário, o servidor irá comparar o CSRF vindo do formulário com o CSRF vindo do cookie
- Os dados do formulário só serão processados caso os dois CSRFs sejam exatamente iguais

O attacker não será capaz de reproduzir o mesmo comportamento através de formulários ocultos porque ele não é capaz de acessar os cookies dos usuários



Cross-Site Request Forgery

- Para usar o CSRF, precisamos instalar o pacote **csurf**

```
$ npm install csurf  
$ npm install -D @types/csurf
```

- Para usar o middleware, precisamos dar importar o módulo e adicioná-lo em nossa aplicação com o método **use**

```
// Arquivo src/server.ts  
import csurf from 'csurf';  
  
this.server.use(express.urlencoded({ extended: false }));  
this.server.use(cookieParser());  
this.server.use(csurf({ cookie: true }));
```

Cross-Site Request Forgery

Minha Loja

Página Principal

Blog RSS

GitHub

Made by T...

Code relea...

Based on B...

Icons. Web for...

Requisição para criação do cookie `_csrf`

Minha Loja

localhost:3333

Elements Console Sources Network

Filter ☐ Invert ☐ Hide data URLs

All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other

☐ Has blocked cookies ☐ Blocked Requests ☐ 3rd-party requests

20 ms 40 ms 60 ms 80 ms 100 ms

Headers Preview Response Initiator Timing Cookies

localhost:3333/main.css

Etag: W/"aed-Cl0QZ0uGk3ftLWtfu5ni6qEjnE0"

Keep-Alive: timeout=5

Set-Cookie: `_csrf=hVEOTFNHkMU3qAA1MOVAgmLd; Path=/`

X-Powered-By: Express

▼ Request Headers ☐ Raw

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7

Accept-Encoding: gzip, deflate, br

Accept-Language: pt,en-US;q=0.9,en;q=0.8

5 requests 765

Error

localhost:3333/departamento/create

ForbiddenError: invalid csrf token

```
at csrf (/home/david/dev/expTsAV2/node_modules/  
at Layer.handle [as handle_request] (/home/davi  
at trim_prefix (/home/david/dev/expTsAV2/node_m  
at /home/david/dev/expTsAV2/node_modules/expres  
at Function.process_params (/home/david/dev/exp
```

A partir deste momento, todo formulário submetido sem o token csrf dará o erro **ForbiddenError: invalid csrf token**

```
at trim_prefix (/home/david/dev/expTsAV2/node_m  
at /home/david/dev/expTsAV2/node_modules/expres
```

Cross-Site Request Forgery

- Para tirar o erro, precisamos devolver o **token csrf** para o servidor após o preenchimento de cada formulário

```
// Arquivo src/controllers/departamento.ts - É preciso  
// enviar o csrf para a view de todos os formulários  
res.render('departamento/create', {  
  csrf: req.csrfToken()  
});
```

```
<!-- Arquivo views/departamento/create.handlebars -->  
...  
<input type="hidden" name="_csrf" value="{{csrf}}">  
<button type="submit">Adicionar</button>  
...
```

Minha Loja

localhost:3333/dept/create

Minha Loja

Adicionar Departamento

Nome do Departamento

O nome precisa conter entre 3 e 50 caracteres.

Sigla do Departamento

Submeter

Exercício: Usar o pacote **csrf** para garantir que todos os formulários da aplicação estejam protegidos de ataques csrf

github
Game

Sessões

- Através de sessões, podemos armazenar informações de estado (variáveis) no lado servidor
- Em vez do browser guardar um cookie por dado, ele guarda apenas um cookie contendo um **id de sessão (connect.sid)**

Requisições do Browser

1 Get index.html HTTP1.1
Host: server.com

3 Get index.html HTTP1.1
Host: server.com
Cookie: connect.sid=6Bh

Requisições de server.com

2 HTTP1.1 200 OK
Set-Cookie: connect.sid=6Bh
... Conteúdo da página...

4 HTTP1.1 200 OK
... Conteúdo da página...

Sessão 6Bh
Dados da sessão no servidor
user: Bruna
itens-carrinho: 3
start: 15:30



Sessões

- Para usarmos as sessões, precisamos instalar um módulo para geração de **valores únicos para os IDs** das sessões
- Uma opção é o módulo **uuid** – Universally Unique Identifier – que é uma implementação do UUID descrito na [RFC 4122](https://tools.ietf.org/html/rfc4122)

```
$ npm install uuid
```

- Os UUIDs são valores de 128 bits que podem ser usados como ID únicos de qualquer coisa em sistemas computacionais
 - Ex: f0221c72-ac30-4796-83f5-fd7a8a4f6b15
- Embora a probabilidade de um UUID ser duplicado não seja nula, ela é próximo o suficiente de zero e pode ser ignorada

Sessões

- Para usarmos as sessões, precisamos instalar um módulo para geração de **valores únicos para os IDs** das sessões
- Uma opção é o módulo **uuid** – Universally Unique Identifier – que é uma implementação do UUID descrito na [RFC 4122](https://tools.ietf.org/html/rfc4122)

```
$ npm install uuid
```

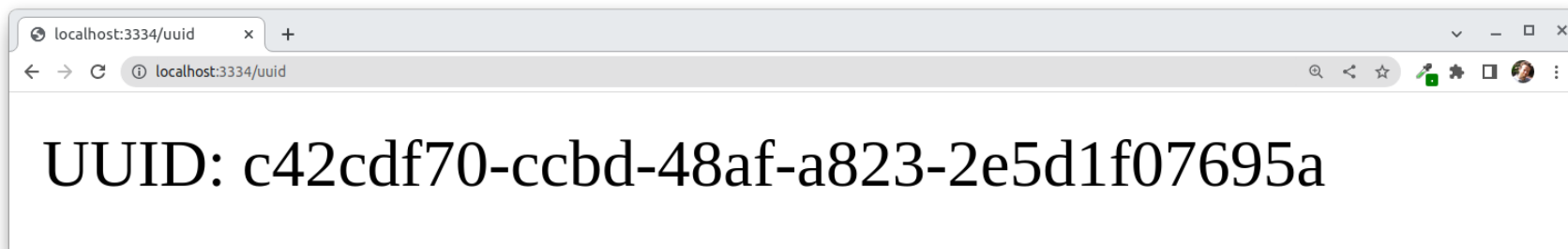
- Os UUIDs são valores de 128 bits que podem ser usados como
Alguns desenvolvedores preferem usar o uuid como **chave primária** de tabelas, ao invés de um ID auto incrementado.
- Embora a probabilidade de um UUID ser duplicado não seja nula, ela é próximo o suficiente de zero e pode ser ignorada

Sessões

- O código abaixo mostra como os UUIDs podem ser gerados a partir do módulo **uuid**

```
// Arquivo src/router/router.ts  
router.get('/uuid', mainController.uuid);
```

```
// Arquivo src/controllers/main.ts  
const uuid = (req, res, next) => {  
  const uniqueId = uuidv4();  
  res.send(`UUID: ${uniqueId}`);  
};
```



Sessões

- O código abaixo mostra como os UUIDs podem ser gerados a partir do módulo **uuid**

```
// Arquivo src/router/router.ts  
router.get('/uuid', mainController.uuid);
```

```
// Arquivo src/controllers/main.ts  
const uuid = (req, res, next) => {  
  const uniqueId = uuidv4();  
  res.send(`UUID: ${uniqueId}`);  
};
```

Existem 5
versões de
UUID, conforme
descrito em
sua RFC

UUID: c42cdf70-ccbd-48af-a823-2e5d1f07695a

Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session  
$ npm install -D @types/express-session
```

- Para usar o middleware, precisamos importá-lo e adicioná-lo em nossa aplicação com o método **use**

```
// Arquivo src/server.ts  
import session from 'express-session';  
  
this.server.use(session({  
  genid: () => uuidv4(), // usamos UUID para gerar os SESSID  
  secret: 'Hi9Cf#mK98',  
  resave: true,  
  saveUninitialized: true,  
}));
```

Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session
$ npm install -D @types/express
```

- Para usar o middleware, precisamos adicioná-lo em nossa aplicação com o seguinte código:

```
// Arquivo src/server.ts
import session from 'express-session'

this.server.use(session({
  genid: () => uuidv4(), // usamos UUID para gerar os SESSID
  secret: 'Hi9Cf#mK98',
  resave: true,
  saveUninitialized: true,
}));
```

Usado para adicionar uma assinatura (similar ao checksum) ao **session.id** enviado para o usuário. Quando o usuário devolve o **session.id**, a assinatura é usada para checar se o **session.id** é válido. Usa uma técnica chamada HMAC.

Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session
$ npm install -D @types/express-session
```

- Para usar o middleware em nossa aplicação

```
// Arquivo src/server.js
import session from 'express-session';

this.server.use(session({
  genid: () => { /* ... */ },
  secret: 'H19Cf#mK98',
  resave: true,
  saveUninitialized: true,
})));
```

Quando **true**, a sessão do usuário é salva a cada requisição, mesmo que os dados da sessão não tenham sido modificados durante a requisição. Isso mantém a sessão ativa, visto que ela pode ser deletada após algum tempo de desuso.

Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session
```

```
$ npm install -D @types/express-session
```

- Para usar o middleware em nossa aplicação

```
// Arquivo src/server.js
import session from 'express-session'

this.server.use(
  session({
    genid: () => {
      // Gerar um ID único para cada sessão
    },
    secret: 'H19CmK98',
    resave: true,
    saveUninitialized: true,
  }));
```

Quando **true**, a sessão do

Quando **true**, força que as sessões não inicializadas sejam salvas no store. Uma sessão não inicializada ocorre quando a sessão é nova e ainda não foi modificada.

Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session
```

```
$ npm install -D @types/express-session
```

Quando **true**, a sessão do usuário é adicionada ao cookie

- Para usar o middleware, é necessário configurar o `session` no arquivo `app.js`

Minha Loja

Página Principal

[Blog](#) [RSS](#) [Back to top](#)

[Twitter](#) [GitHub](#) [API](#)

[Donate](#)

Application

- Manifest
- Service Workers
- Storage

Storage

- Local Storage

Filter

Name	Value	Domain	Path	Expires	Secure	HttpOnly	SameSite	Session	Priority
connect.sid	s...	localhost	/	9/1/2024	✓	✓	Strict	✓	M
csrf	m...	localhost	/	2/1/2024	✓	✓	Strict	✓	M

Cookie Value ☐ Show URL-decoded

s%3Af41f6927-e40e-40ff-bb6b-2b17939ad82a.3buE044L9kA%2BQeISxObXVfhXmooFzyjraHJMoNb3Pyk

Cadastro de Usuários

- O cadastro de usuário envolve, dentre outras coisas, a geração de uma senha criptografada e seu registro no banco de dados
- O formulário de cadastro de usuário possui os seguintes campos:
 - **Nome completo** – input text
 - **Endereco de e-mail** – input e-mail
 - **Seu curso na UFAM** – select
 - **Escolha uma senha de acesso** – input password
 - **Confirme sua senha** – input password
 - **Eu li e concordo com os termos de servico** – radio box

- O cadastro o de um senha
- O formulário campos:
 - Nome comp
 - Endereco d
 - Seu curso n
 - Escolha um
 - Confirme su
 - Eu li e conc

 Chess App

Criar conta

Deseja criar uma conta? Basta preencher o formulário abaixo.

Nome Completo

Endereço de E-mail

Seu curso na UFAM

Selecione seu curso ▾

Escolha uma senha de acesso

Confirme sua senha

☐ Eu li e concordo com os [termos de serviço](#).

Criar conta

Já possui uma conta? [Faça o login](#).

as, a geração
co de dados
guintes

box

- O cadastro o de um senha
- O formulário de cadastro possui os seguintes campos:

The screenshot shows the 'Criar conta' (Create account) page of the Chess App. At the top, there is a blue header with a white chess knight icon and the text 'Chess App'. Below the header, the title 'Criar conta' is centered in a large white font. Underneath the title, a subtitle in a smaller white font reads 'Deseja criar uma conta? Basta preencher o formulário abaixo.' The form itself is a dark gray rectangle with several white input fields. The first field is labeled 'Nome Completo' and contains the text 'Jorge A. Menezes'. Below this, there are two more fields: 'Escolha uma senha de acesso' and 'Confirme sua senha', both containing masked text represented by dots. Under the password fields, there is a checkbox labeled 'Eu li e concordo com os termos de serviço.' and a blue button labeled 'Criar conta'. At the bottom of the form, there is a link that says 'Já possui uma conta? Faça o login.'

Note que será preciso verificar se 1) o usuário digitou a mesma string nos campos senha e confirmação de senha e; 2) se o usuário concordou com os termos de serviço

- Escolha um
- Confirme su
- Eu li e conc

- O cadastro o de um senha
- O formulário de cadastro tem os seguintes campos:

– Note que será preciso verificar se 1) o usuário digitou a mesma
 – Para fazer isso, uma opção seria usar o mecanismo de validação
 – do sequelize (nesse caso, a validação seria feita no servidor).
 – Outra opção é desenvolver um código javascript para validação
 – desses campos no lado cliente.

- Confirme sua
- Eu li e concordo com os

Criptografando as senhas

- Após a submissão do formulário, é necessário criptografar a senha antes de salvá-la no banco de dados
- Mas por quê? Por que não guardar a senha crua?
 - Todas as pessoas com acesso ao banco poderiam ver a senha
 - Os usuários frequentemente **usam a mesma senha** em vários sites
 - A senha iria aparecer nos **backups** do banco
 - Se o banco estiver na **cloud**, as senhas ficariam expostas na web
 - As senhas ficariam expostas a **ataques de SQL-injection**

Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela **Estado**

estado	capital
Rio de Janeiro	Rio de Janeiro
Amazonas	Manaus
Minas Gerais	Belo Horizonte
Ceará	Fortaleza

Tabela **Usuario**

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

Busca da capital pelo estado

Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela **Estado**

estado	capital
Rio de Janeiro	Rio de Janeiro
Amazonas	Manaus
Minas Gerais	Belo Horizonte
Ceará	Fortaleza

Tabela **Usuario**

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

Busca da capital pelo estado


```
SELECT estado, capital FROM estado  
WHERE estado = 'Amazonas'
```

estado	capital
Amazonas	Manaus

Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela **Estado**

estado	capital
Rio de Janeiro	Rio de Janeiro
Amazonas	Manaus
Minas Gerais	Belo Horizonte
Ceará	Fortaleza

Tabela **Usuario**

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

Busca da capital pelo estado

' UNION SELECT login, senha FROM Usuario WHERE login != '

Submit



SELECT estado, capital **FROM** Estado
WHERE estado = ' UNION SELECT login,
senha **FROM** Usuario **WHERE** login != '

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela **Estado**

estado	capital
Rio de Janeiro	Rio de Janeiro
A	
M	
Ceará	Fortaleza

Tabela **Usuario**

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

Os ataques de SQL-Injection só são possíveis se o invasor conseguir incluir caracteres como ' (aspas simples), " (aspas duplas) ou \b (caracter de backspace) nos inputs dos formulários.

Busca da capital pelo estado

' UNION SELECT login, senha FROM Usuario WHERE login != '

Submit

SELECT estado, capital **FROM** estado
WHERE estado = ' UNION SELECT login,
senha **FROM** Usuario **WHERE** login != '

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela **Estado**

estado	capital
Rio de Janeiro	Rio de Janeiro
A	
M	
C	

Tabela **Usuario**

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

O Sequelize resolve este problema adicionando um caracter de **escape** a tais elementos. Por exemplo, o caracter ' vira \', " vira \" e \b vira \\b. Vide [função de escape de Sequelize](#). Mesmo assim, é sempre saudável fazer validação dos inputs.

Busca da capital pelo estado

' UNION SELECT login, senha FROM Usuario WHERE login != '

Submit

SELECT estado, capital **FROM** estado
WHERE estado = ' UNION SELECT login,
senha **FROM** Usuario **WHERE** login != '

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

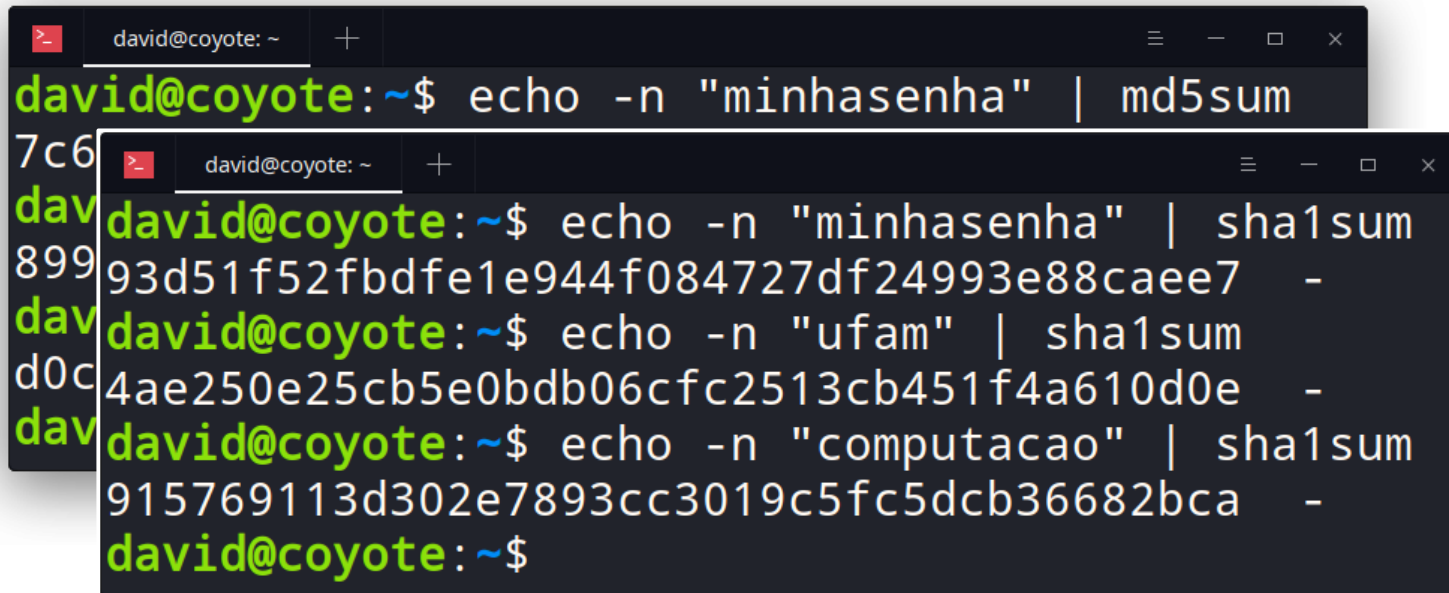
Funções HASH

- Uma função Hash é um algoritmo que transforma um bloco de dados qualquer em série de caracteres de comprimento fixo
 - Ex: md5, sha1, sha256, etc
- São funções de uma **mão-única**, isto é, não é possível recuperar o bloco de dados original a partir do hash gerado

```
david@coyote: ~  
david@coyote:~$ echo -n "minhasenha" | md5sum  
7c67e713a4b4139702de1a4fac672344 -  
david@coyote:~$ echo -n "ufam" | md5sum  
8996fe161805927c27a92fae2ca238d2 -  
david@coyote:~$ echo -n "computacao" | md5sum  
d0cc5ed8aecb1898032c48af57057b9f -  
david@coyote:~$
```

Funções HASH

- Uma função Hash é um algoritmo que transforma um bloco de dados qualquer em série de caracteres de comprimento fixo
 - Ex: md5, sha1, sha256, etc
- São funções de uma **mão-única**, isto é, não é possível recuperar o bloco de dados original a partir do hash gerado



```
david@coyote: ~$ echo -n "minhasenha" | md5sum
7c6
david@coyote: ~$ echo -n "minhasenha" | sha1sum
899 93d51f52fbdfef1e944f084727df24993e88caee7 -
david@coyote: ~$ echo -n "ufam" | sha1sum
d0c 4ae250e25cb5e0bdb06cfc2513cb451f4a610d0e -
david@coyote: ~$ echo -n "computacao" | sha1sum
915769113d302e7893cc3019c5fc5dcb36682bca -
david@coyote: ~$
```

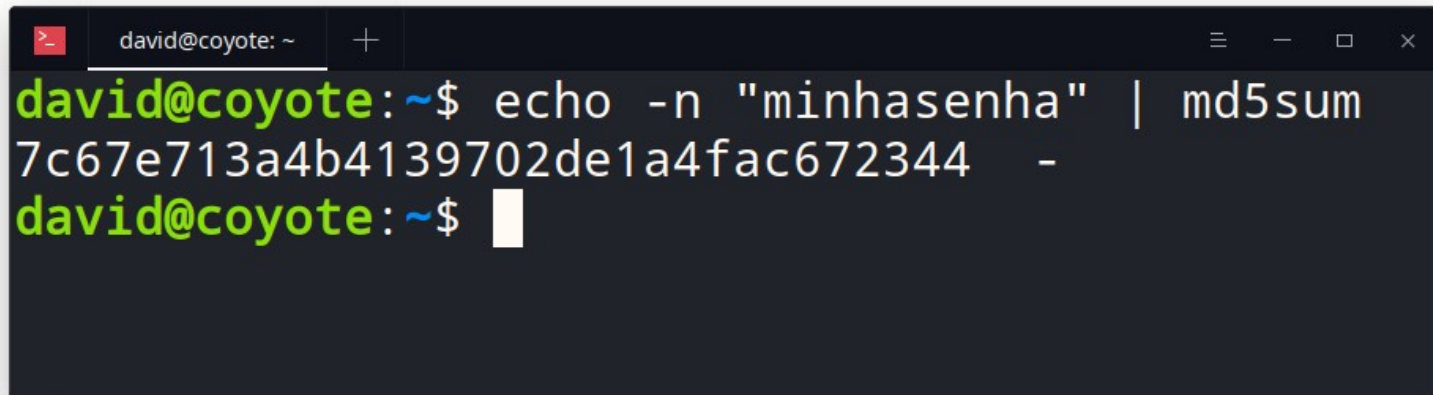
Funções HASH

- Uma função Hash é um algoritmo que transforma um bloco de dados qualquer em série de caracteres de comprimento fixo
- Os comandos **md5sum** e **sha1sum** são instalados por padrão em sistemas UNIX, GNU/Linux e BSD. Eles são usados para verificar a integridade de dados transmitidos através da Web. Para maiores informações, vide [Soma de Verificação \(Checksum\)](#)

```
david@coyote: ~$ echo -n "minhasenha" | md5sum
7c6
david@coyote: ~$ echo -n "minhasenha" | sha1sum
899 93d51f52fbdfef1e944f084727df24993e88caee7 -
david@coyote: ~$ echo -n "ufam" | sha1sum
d0c 4ae250e25cb5e0bdb06cfc2513cb451f4a610d0e -
david@coyote: ~$ echo -n "computacao" | sha1sum
915769113d302e7893cc3019c5fc5dcb36682bca -
david@coyote: ~$
```

Funções HASH

- Considere um sistema que usa a função hash MD5 para criptografar as senhas de seus usuários
 - Ex: se a senha de um usuário é **minhasenha**, a string armazenada no banco será **7c67e713a4b4139702de1a4fac672344**
 - Quando o usuário tentar logar no sistema, o servidor irá comparar o md5 da senha informada com a string armazenada no banco

A terminal window with a dark background. The title bar shows 'david@coyote: ~' and standard window controls. The prompt is 'david@coyote:~\$'. The command 'echo -n "minhasenha" | md5sum' is entered. The output '7c67e713a4b4139702de1a4fac672344 -' is displayed. The prompt 'david@coyote:~\$' is shown again with a cursor.

```
david@coyote:~$ echo -n "minhasenha" | md5sum
7c67e713a4b4139702de1a4fac672344 -
david@coyote:~$
```

Funções HASH

- Considere um sistema que usa a função hash MD5 para criptografar as senhas de seus usuários
 - Ex: se a senha de um usuário é **minhasenha**, a string armazenada no banco será **7c67e713a4b4139702de1a4fac672344**
 - Quando o usuário tentar logar no sistema, o servidor irá comparar o md5 da senha informada com a string armazenada no banco
- Note que **ninguém** poderá descobrir a senha do usuário caso acesso a tabela Usuário do banco de dados

```
david@coyote:~$ echo -n "minhasenha" | md5sum
7c67e713a4b4139702de1a4fac672344 -
david@coyote:~$
```

Funções HASH

- Considere um sistema que usa a função hash MD5 para criptografar as senhas de seus usuários
 - Ex: se a senha de um usuário é **minhasenha**, a string armazenada no banco será **7c67e713a4b4139702de1a4fac672344**
 - Quando o usuário tentar logar no sistema, o servidor irá comparar o md5 da senha informada com a string armazenada no banco
- Note que **ninguém** poderá descobrir a senha do usuário caso a
- Será mesmo? Então vamos no Google e inserir a string hash **7c67e713a4b4139702de1a4fac672344** no campo de busca

```
7c67e713a4b4139702de1a4fac672344 -  
david@coyote:~$
```


Funções HASH

- Considere a criptografia

- Ex: se a senha não bater

- Quando o md5

Note a

Será
7c67

7c67e
davic

7c67e713a4b4139702de1a4fac672344

Todas Maps Vídeos Imagens Shopping Mais Configurações

Aproximadamente 43 resultados (0,27 segundos)

7c67e713a4b4139702de1a4fac672344 - Hash Toolkit
[https://hashtoolkit.com > reverse-md5-hash > 7c67e713a4b4139702de1a4fac672344](https://hashtoolkit.com/reverse-md5-hash/7c67e713a4b4139702de1a4fac672344) - Traduzir esta página
Decrypt md5 Hash Results for: 7c67e713a4b4139702de1a4fac672344 ... md5, 7c67e713a4b4139702de1a4fac672344, minhasenha ...

Hash Md5: 7c67e713a4b4139702de1a4fac672344 - MD5Hashing.net
[https://md5hashing.net > hash > 7c67e713a4b4139702de1a4fac672344](https://md5hashing.net/hash/7c67e713a4b4139702de1a4fac672344)
Decoded hash Md5: 7c67e713a4b4139702de1a4fac672344: minhasenha.

Usando o md5 - Recursos do PHP
[recursosdophp.blogspot.com > normal-0-21-false-false-false-pt-br-x](https://recursosdophp.blogspot.com/2021/02/normal-0-21-false-false-false-pt-br-x) ▾
md5("minhasenha") = "7c67e713a4b4139702de1a4fac672344". Com isso se você modificar o valor de "\$senha" ele vai dar senha Invalida. Bom, isso até mais .



Funções HASH

Best MD5 Password Dec x

Secure | <https://hashtoolkit.com/reverse-md5-hash/7c67e713a4b4139702de1a4fac672344>

Hash Toolkit

Search in 15,786,077,212 decrypted md5 / sha1 hashes.

Hash:

Decrypt md5 Hash Results for: **7c67e713a4b4139702de1a4fac672344**

Algorithm	Hash	Decrypted
md5	7c67e713a4b4139702de1a4fac672344	minhasenha

recursosdophp.blogspot.com > normal-0-21-false-false-false-pt-br-x ▾

md5("minhasenha") = "7c67e713a4b4139702de1a4fac672344". Com isso se você mudar o valor de "\$senha" ele vai dar senha Invalida. Bom, isso até mais .



Funções HASH

Best MD5 Password Dec x

Secure | <https://hashtoolkit.com/reverse-md5-hash/7c67e713a4b4139702de1a4fac672344>

Hash Toolkit

Search in 15,786,077,212 decrypted md5 / sha1 hashes.

Sistemas como o **Hash Toolkit** usam tabelas contendo bilhões de valores de hash pré-calculados. Essas tabelas normalmente são chamadas de **rainbow tables**.

Decrypt md5 Hash Results for: **7c67e713a4b4139702de1a4fac672344**

Algorithm	Hash	Decrypted
md5	7c67e713a4b4139702de1a4fac672344	minhasenha

recursosdophp.blogspot.com > normal-0-21-false-false-false-pt-br-x ▾

md5("minhasenha") = "7c67e713a4b4139702de1a4fac672344". Com isso se você usar o valor de "\$senha" ele vai dar senha Invalida. Bom, isso até mais .



Salt



- O **salt** é uma estratégia utilizada para evitar que duas senhas idênticas produzam **hashes** idênticos
- A ideia é concatenar a senha original do usuário com uma sequência adicional de caracteres aleatórios, chamada salt
- Para exemplificar, podemos aplicar o salt **Us8#upK12MjsM** à senha original do usuário, **minhasenha**

```

david@coyote: ~$ echo -n "Us8#upK12MjsMminhasenha" | md5sum
d3eeb71a83744a4bbaa3ce41be87a292 -
david@coyote: ~$
```

Salt



- O **salt** é uma estratégia utilizada para evitar que duas senhas idênticas produzam **hashes** idênticos
- A ideia é concatenar a senha original do usuário com uma

d3eeb71a83744a4bbaa3ce41be87a292

Todas Maps Vídeos Imagens Shopping Mais Configurações

Sua pesquisa - **d3eeb71a83744a4bbaa3ce41be87a292** - não encontrou nenhum documento correspondente.

Sugestões:

- Certifique-se de que todas as palavras estejam escritas corretamente.
- Tente palavras-chave diferentes.
- Tente palavras-chave mais genéricas.

Google

Salt



- O **salt** é uma estratégia utilizada para evitar que duas senhas idênticas produzam **hashes** idênticos
- A ideia é concatenar a senha original do usuário com uma

De uma forma geral, o salt é gerado aleatoriamente para cada usuário e armazenado no banco junto com o hash MD5 resultante:

Us8#upK12MjsMd3eeb71a83744a4bbaa3ce41be87a292

Sua pesquisa - **d3eeb71a83744a4bbaa3ce41be87a292** - não encontrou nenhum documento correspondente.

Sugestões:

- Certifique-se de que todas as palavras estejam escritas corretamente.
- Tente palavras-chave diferentes.
- Tente palavras-chave mais genéricas.

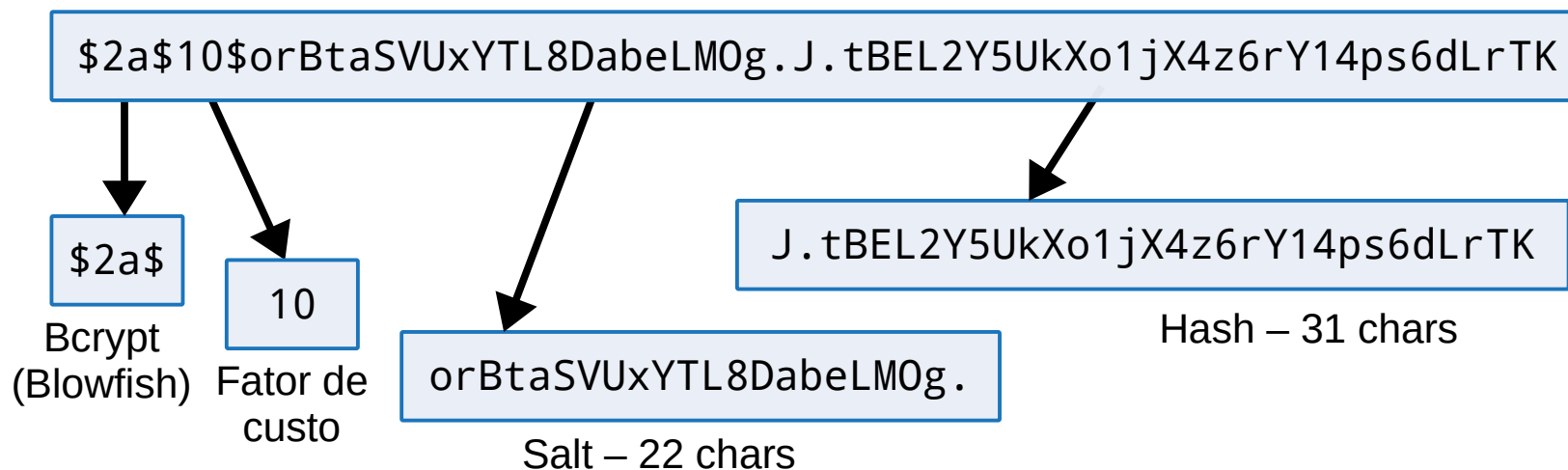


O módulo bcrypt

- O módulo **bcrypt** é uma boa opção para geração de senhas, pois incorpora uma função hash à uma estratégia de salt

```
$ npm install bcryptjs
```

- O bcrypt é um **algoritmo de hash** usado para geração de senhas em sistemas como OpenBSD e algumas distribuições linux



O módulo bcrypt

- Para gerar uma senha com salt podemos usar o código abaixo

```
// Arquivo api/controllers/main.js
const bcrypt = require('bcryptjs');

// Dentro da função signup
bcrypt.genSalt(rounds, function(err, salt) {
  bcrypt.hash(req.body.senha, salt, async(err, hash) => {
    await User.create({
      nome: req.body.nome,
      email: req.body.email,
      senha: hash,
      cursoId: req.body.cursoId
    });
  });
});
```


O módulo bcrypt

- Para gerar uma senha com salt, usar o código abaixo

```
// Arquivo api/controllers
const bcrypt = require('bcrypt')

// Dentro da função signup
bcrypt.genSalt(rounds, function(err, salt) {
  bcrypt.hash(req.body.senha, salt, async(err, hash) => {
    await User.create({
      nome: req.body.nome,
      email: req.body.email,
      senha: hash,
      cursoId: req.body.cursoId
    });
  });
});
```

Número de rounds para geração do hash

Senha informada pelo usuário

O módulo bcrypt

- Para gerar uma senha com salt, usar o código abaixo

```
// Arquivo api/controllers
const bcrypt = require('bcrypt')

// Dentro da função signup
bcrypt.genSalt(rounds, function(err, salt) {
  bcrypt.hash(req.body.senha, salt, async(err, hash) => {
    await User.create({

```

Número de rounds para geração do hash

nome	email	senha	id_curso
David Fernandes	david@icomp.ufam.edu.br	\$2a\$10\$LDsc/xMa91HiUY03KiQxVuZUJCbN0CNcA6F/k9vPH9H/NII/MTNqO	2

```
    senha: hash,
    cursoId: req.body.cursoId
  });
});
});
});
```

Informada pelo usuário

Login de Usuários

- Da mesma forma que o **signup**, a action responsável pelo **login** pode ser colocada no **controlador main**
- Na action login, usamos o método **bcrypt.compare()** para verificar se a senha digitada pelo usuário está correta ou não

```
var user= await User.findOne({where:{email:req.body.email}});  
  
if (user) {  
  bcrypt.compare(req.body.senha, user.senha, (err, ok) => {  
    if (ok) {  
      req.session.uid = user.id;  
      res.redirect('/');  
    } else {  
      res.render('main/login', {  
        csrf: req.csrfToken()  
      });  
    }  
  });  
}
```

Note que, caso o e-mail e senha do usuário estejam corretos, a **variável de sessão uid** é criada



Chess App

Criar conta

Deseja criar uma conta? Basta preencher o formulário abaixo.

Nome Completo

Jorge A. Me

Endereço de E

jorge@exam

Seu curso na U

Selecione seu curso

Escolha uma senha de acesso

.....

Exercício 11: Criar o formulário onde os usuários poderão se cadastrar para usar a aplicação. Após um usuário se cadastrar, sua senha deverá ser criptografada usando as diretrizes apresentadas nos slides.

github
Game

Logout de Usuários

- O controlador main também deverá conter uma action **logout**, que será usada para **encerrar a sessão** do usuário

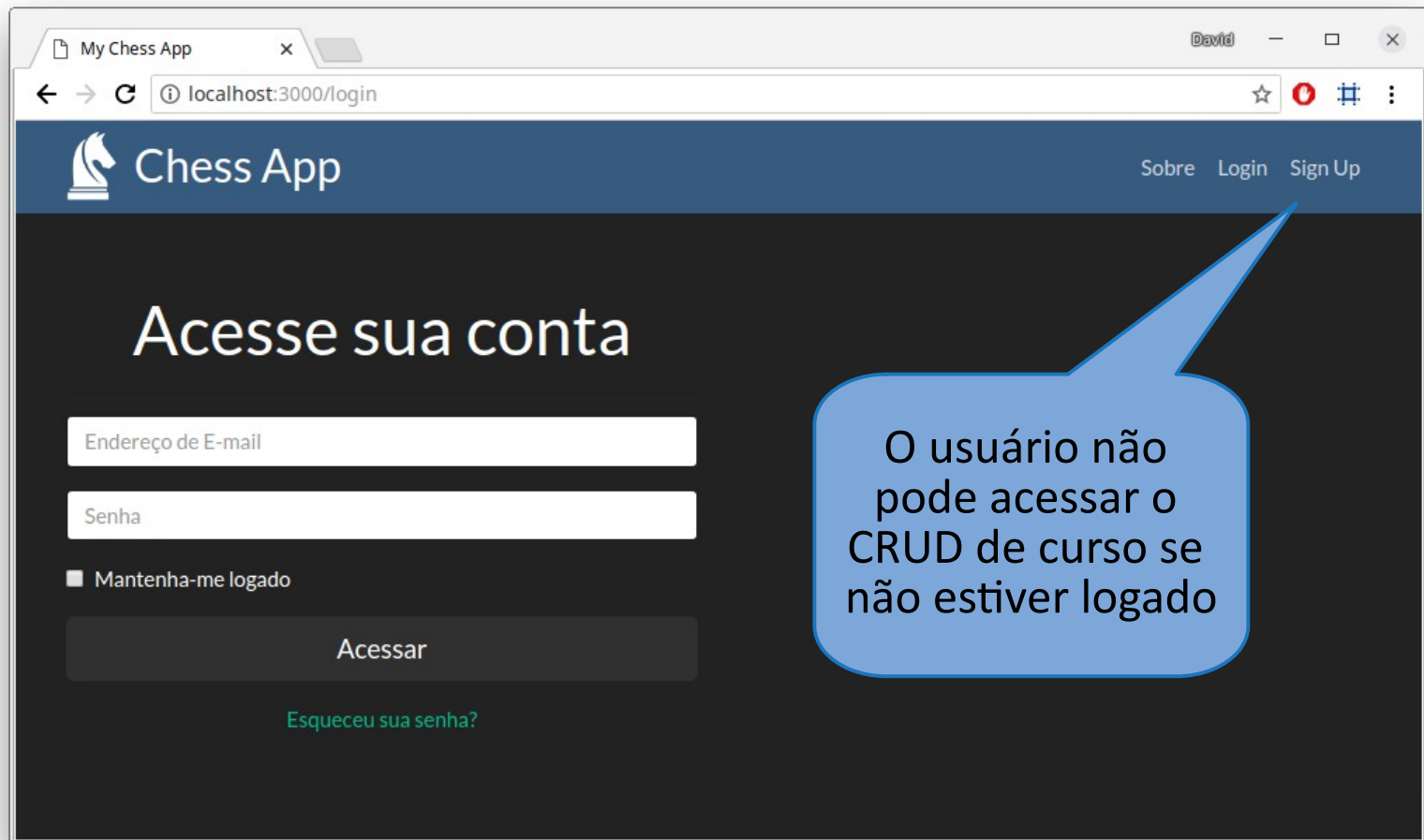


- Para encerrar a sessão, destruindo todas as suas variáveis, usamos o método **req.session.destroy()**

```
const logout = (req, res) => {  
  req.session.destroy(function (err) {  
    if (err) res.send(err);  
    else res.redirect("/")  
  });  
}
```

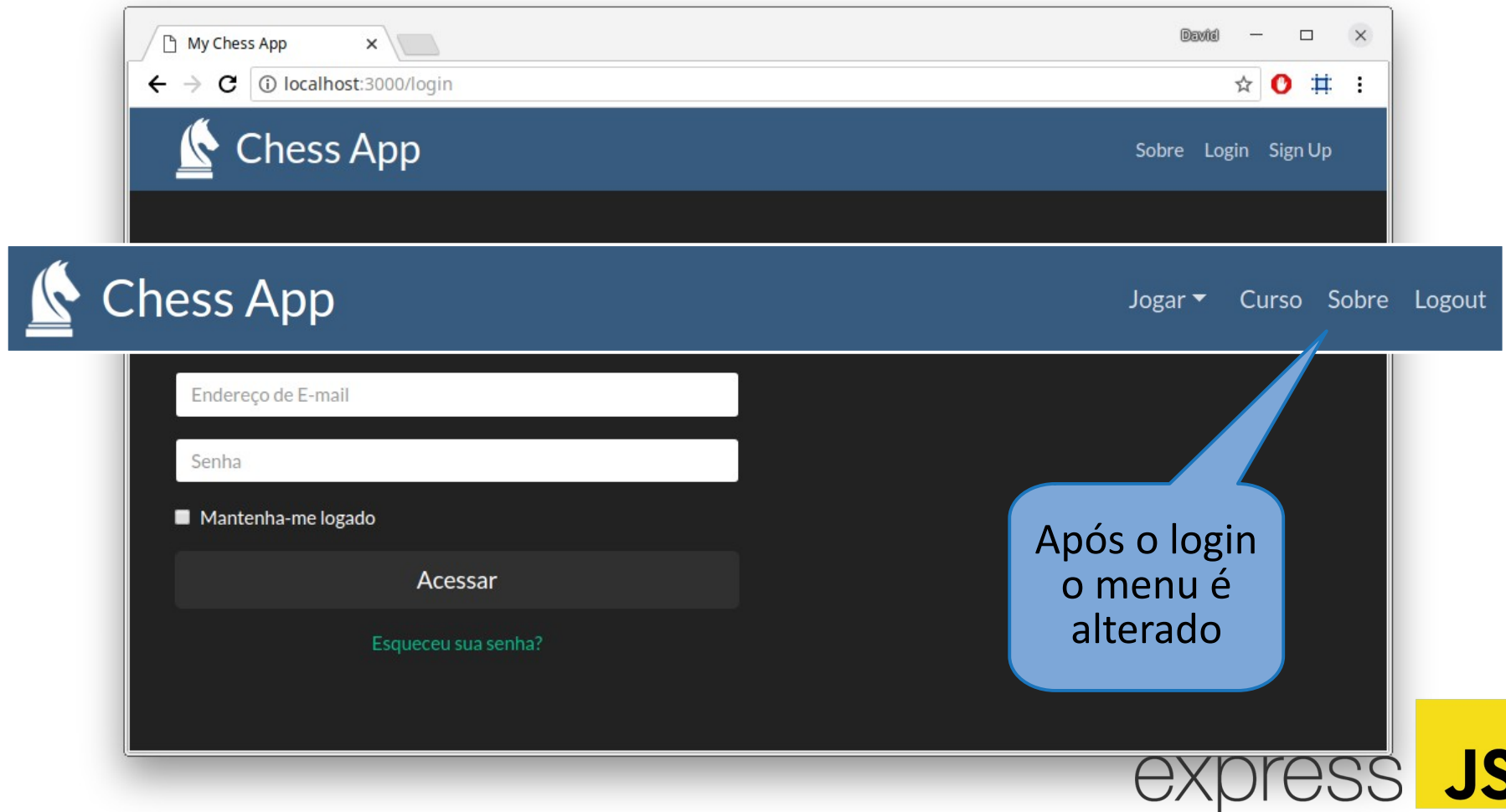
Adaptação do menu superior

- Note que o **menu superior** da aplicação é alterado a partir do momento em que o usuário efetua o login no sistema



Adaptação do menu superior

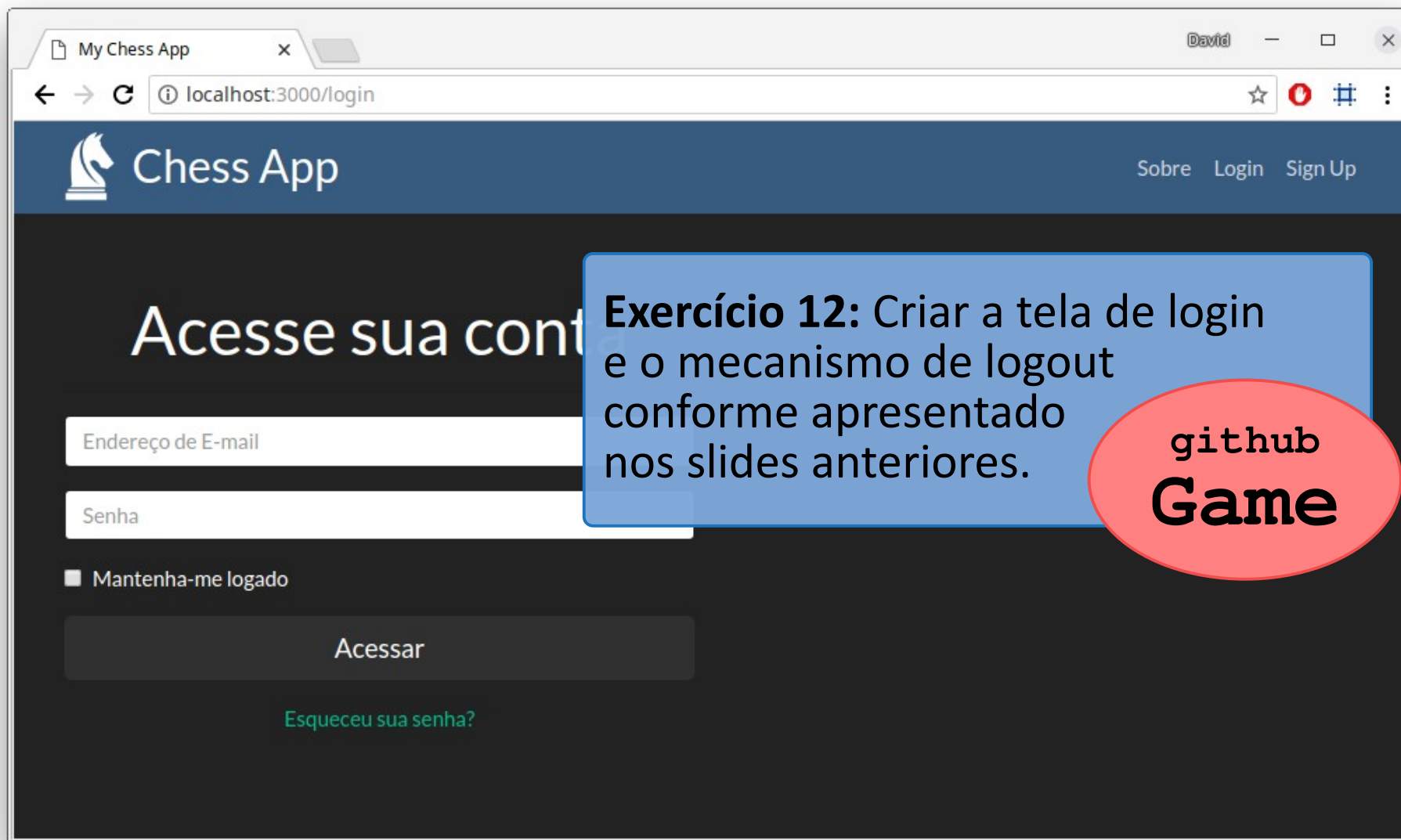
- Note que o **menu superior** da aplicação é alterado a partir do momento em que o usuário efetua o login no sistema





Chess App

Jogar ▾ Curso Sobre Logout



Exercício 12: Criar a tela de login e o mecanismo de logout conforme apresentado nos slides anteriores.

github
Game