GEX613 – Programação II

Autenticação & Autorização

1100/1101 – CIÊNCIA DA COMPUTAÇÃO

Prof. Dr. Giancarlo Salton

- ***Autenticação*** é o processo de provar que você é quem diz ser.

- ***Autorização*** é o ato de conceder a uma parte autenticada a permissão para fazer algo.
  - ✔ Ele especifica quais dados você tem permissão para acessar e o que pode fazer com esses dados.
  - ✔ Às vezes, a autorização é abreviada para **AuthZ**.

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

Autenticação

O que é autenticação?

- Em uma aplicação web, autenticação refere-se ao processo de verificar a identidade de um usuário para garantir que ele seja quem alega ser.

- Isso geralmente envolve a coleta e verificação de credenciais, como nome de usuário e senha, para permitir o acesso seguro a recursos e funcionalidades específicas da aplicação.

- A autenticação é fundamental para proteger informações sensíveis e restringir o acesso apenas a usuários autorizados.

O que é sessão?

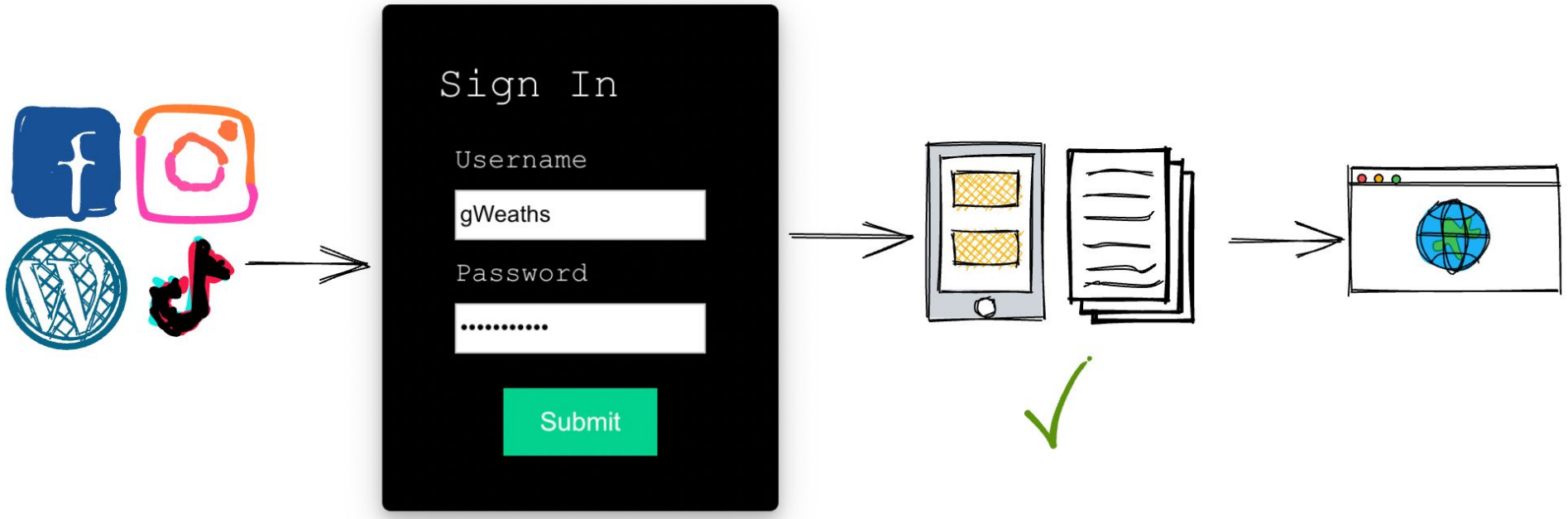
- Uma sessão em uma aplicação web é um estado de interação entre um cliente e um servidor que é armazenado temporariamente no servidor e, geralmente, associado a um usuário específico.

- As sessões são usadas para armazenar informações específicas do usuário entre as requisições, permitindo a persistência de dados durante a interação do usuário com a aplicação.

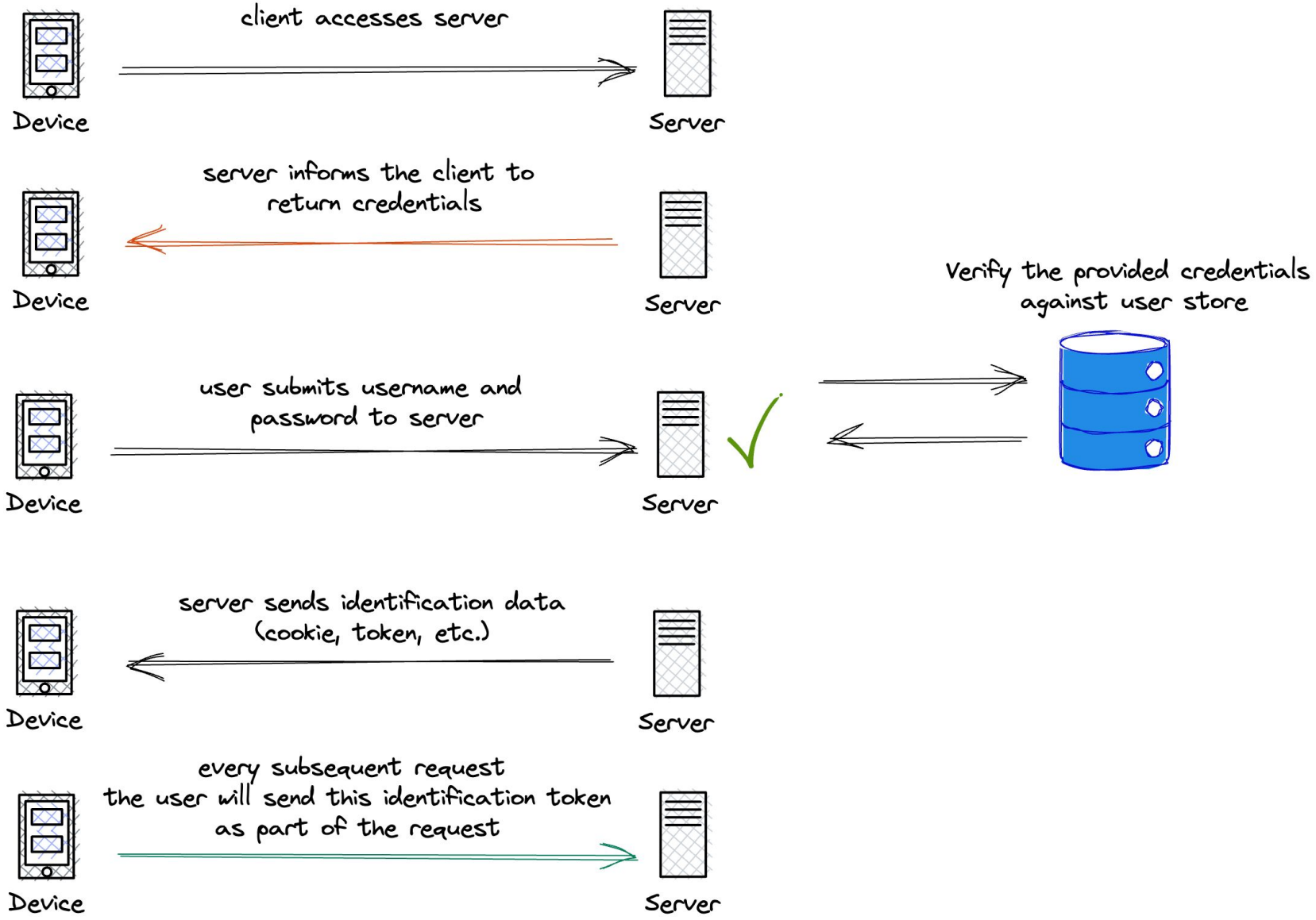
- ***Autenticação*** é o processo de provar que você é quem diz ser.
  - Isso é obtido pela verificação da identidade de uma pessoa ou de um dispositivo.
  - Às vezes, a autenticação é abreviada para ***AuthN***

- A autenticação pode ser aprovada por credenciais de login, como:
  - ✔ Nomes de usuário e senhas
  - ✔ Tokens de acesso e códigos de uso único
  - ✔ Perguntas de segurança
  - ✔ Aplicativos de autenticação associados a um número de telefone ou e-mail

Processo de Autenticação



Processo de Autenticação com JWT

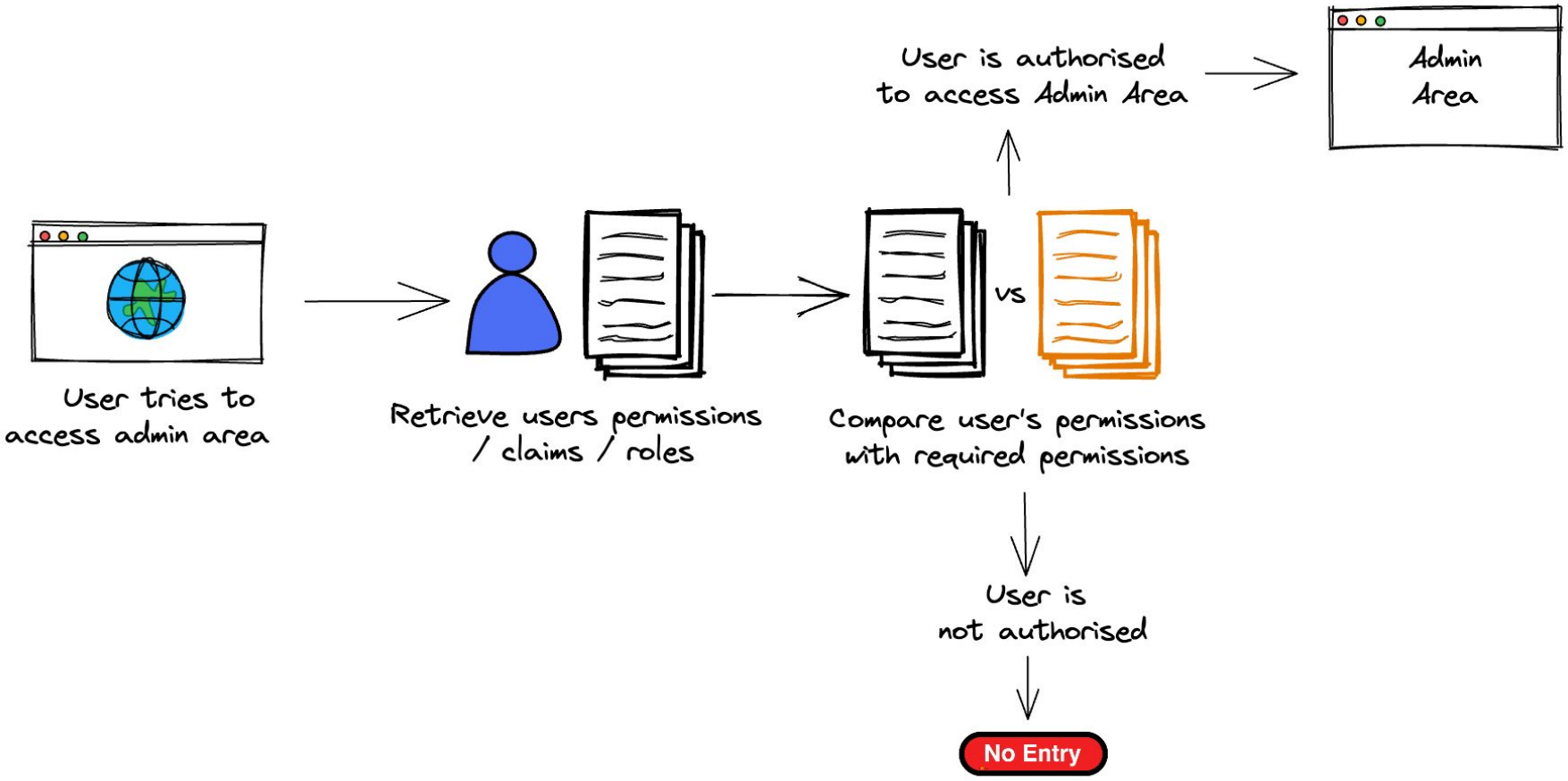


UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

Autorização

- ***Autorização*** é o ato de conceder a uma parte autenticada a permissão para fazer algo.
  - ✔ Ele especifica quais dados você tem permissão para acessar e o que pode fazer com esses dados.
  - ✔ Às vezes, a autorização é abreviada para **AuthZ**.

Processo de Autorização



UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

Backend

```
const session = require("express-session");
const passport = require("passport");
const LocalStrategy = require("passport-local");
const jwt = require("jsonwebtoken");
const bcrypt = require("bcrypt");
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;
```

- Middleware é um software que atua como um intermediário entre diferentes componentes ou sistemas.
- O express-session é um middleware para o framework Node.js chamado Express, que é amplamente utilizado para criar aplicativos web e APIs.
- Em aplicações web, um middleware é uma função que tem acesso ao objeto de requisição (req), objeto de resposta (res), e à próxima função no ciclo de requisição-resposta do servidor.
- O express-session é especificamente projetado para gerenciar sessões em aplicações Express.

- https://expressjs.com/en/resources/middleware/session.html

express-session

npm v1.17.3 | downloads 13.4M/month | ci unknown | coverage 100%

Installation

This is a Node.js module available through the npm registry. Installation is done using the `npm install` command:

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

API

```
var session = require('express-session')
```

session(options)

Create a session middleware with the given options.

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

- Permite armazenar dados associados a uma sessão específica. Esses dados podem ser acessados e modificados ao longo do ciclo de vida da sessão.

- Facilita o uso de cookies para rastrear a identificação da sessão do usuário. Ele cria e envia um cookie contendo um identificador de sessão exclusivo para o navegador do cliente, permitindo que a aplicação identifique o usuário em solicitações subsequentes.

- Ele fornece uma variedade de opções de configuração para ajustar o comportamento das sessões, como tempo de expiração, armazenamento personalizado e opções de segurança.

- Ele ajuda a proteger contra ataques de sessão, como ataques de fixação de sessão, invalidando e regenerando automaticamente as identificações de sessão.

- **secret**: É uma string usada para assinar o cookie de sessão, garantindo que o conteúdo do cookie não seja facilmente manipulado.
- **resave**: Indica se a sessão deve ser salva no armazenamento mesmo se não houve alterações durante a solicitação.
- **saveUninitialized**: Indica se uma sessão não inicializada (não modificada) deve ser salva.
- **cookie**: Indica que o cookie de sessão só será enviado pelo navegador se a conexão estiver usando HTTPS, pois o HTTPS criptografa a comunicação entre o cliente e o servidor.

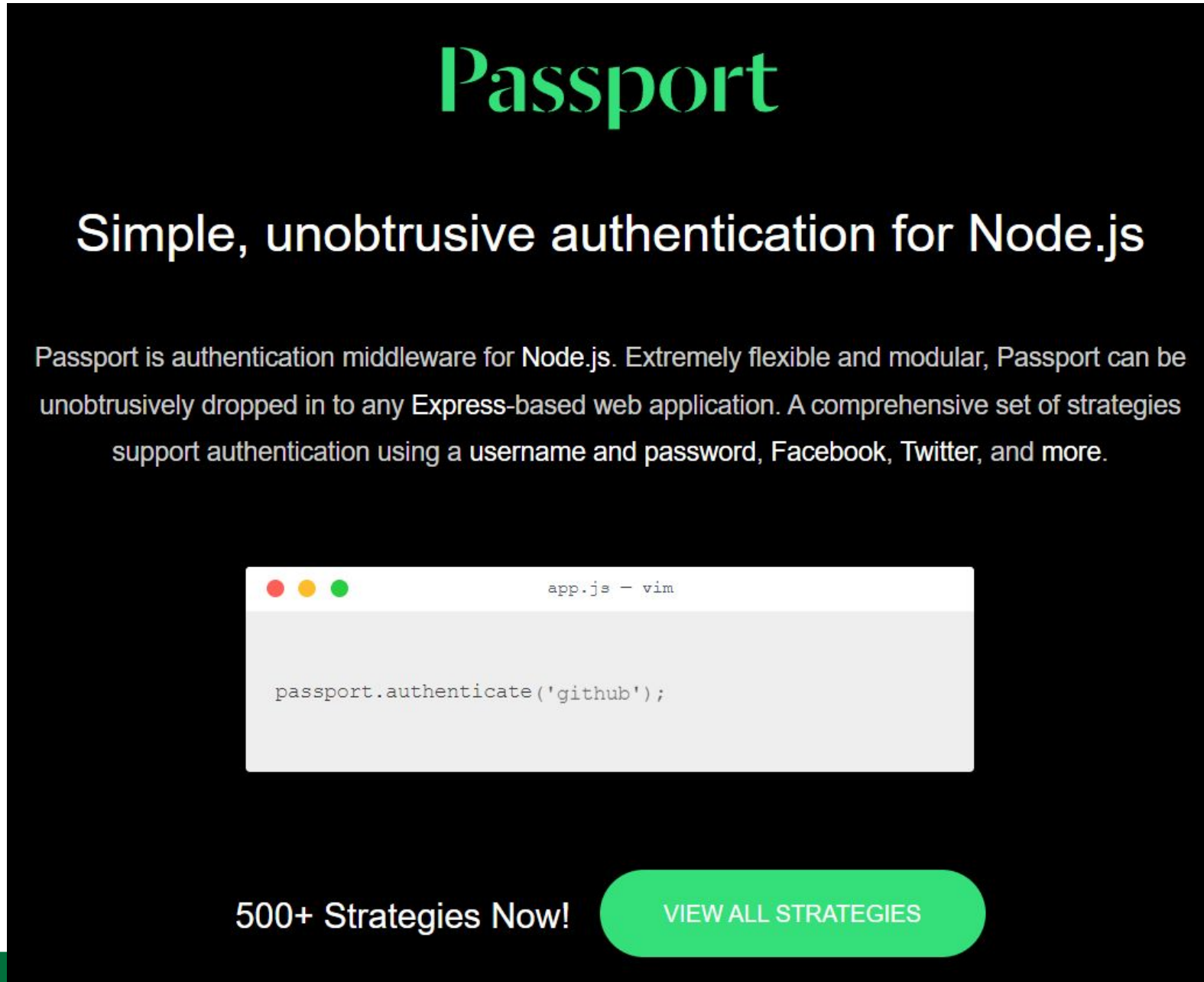
```
const session = require("express-session");
app.use(
    session({
        secret: '"alguma_frase_muito_doida_pra_servir_de_SECRET',
        resave: false,
        saveUninitialized: false,
        cookie: { secure: true },
    }),
);
```

Passport JS

- O Passport.js é um middleware de autenticação para Node.js que simplifica o processo de autenticação em aplicações web.
- Oferece uma maneira flexível e modular de integrar autenticação em uma aplicação baseada em Express.
- Permitindo a utilização de diversas estratégias de autenticação, como usuário/senha, OpenID, GitHub, Facebook, LinkedIn, Twitter, JWT (JSON Web Token), entre outras.

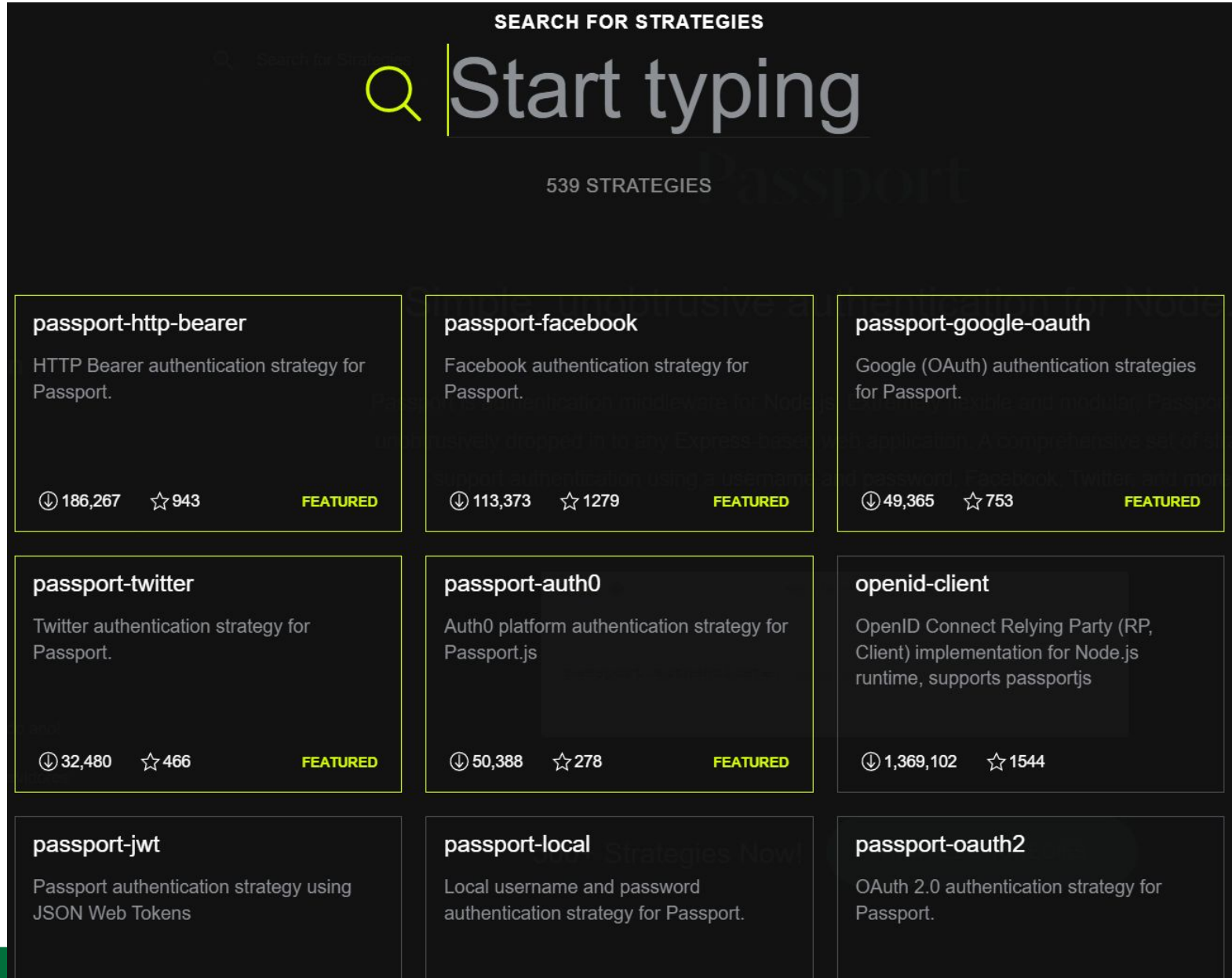
Passport JS

- https://www.passportjs.org/



Passport JS

- O Passport utiliza o conceito de estratégias de autenticação.
- Cada estratégia é um módulo independente que trata de um método específico de autenticação.



- O Passport é projetado para ser utilizado como middleware em aplicações Express.js. Isso facilita a integração com o framework e permite que o Passport gerencie o processo de autenticação durante as requisições HTTP.

- O Passport é altamente configurável e modular. Isso significa que você pode escolher as estratégias que melhor se adequam às necessidades da sua aplicação e combiná-las da maneira que desejar.

```
const passport = require("passport");
const LocalStrategy = require("passport-local");
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;

app.use(passport.initialize());
app.use(passport.session());
```

- O Passport simplifica a implementação em comparação com a criação manual. Ele lida com muitos detalhes, como a comunicação entre o cliente e o servidor, a validação de credenciais, a geração e verificação de tokens, etc.

- Ele também fornece métodos de serialização e desserialização, permitindo que você controle como os dados do usuário são armazenados na sessão.

```
passport.serializeUser(function (user, cb) {
    process.nextTick(function () {
        return cb(null, {
            user_id: user.user_id,
            username: user.user_id,
        });
    });
});
```

```
passport.deserializeUser(function (user, cb) {
    process.nextTick(function () {
        return cb(null, user);
    });
});
```

Passport JS

passport-local

build unknown | coverage 98% | maintainability C | 404 badge not found

Passport strategy for authenticating with a username and password.

This module lets you authenticate using a username and password in your Node.js applications. By plugging into Passport, local authentication can be easily and unobtrusively integrated into any application or framework that supports Connect-style middleware, including Express.

Install

```
$ npm install passport-local
```

Usage

Configure Strategy

The local authentication strategy authenticates users using a username and password. The strategy requires a `verify` callback, which accepts these credentials and calls `done` providing a user.

```
passport.use(new LocalStrategy(
  function(username, password, done) {
    User.findOne({ username: username }, function (err, user) {
      if (err) { return done(err); }
      if (!user) { return done(null, false); }
      if (!user.verifyPassword(password)) { return done(null, false); }
      return done(null, user);
    });
  }
));
```

Passport - Local Strategy

```
npm install passport-local
```

```
yarn add passport-local
```

```
passport.use(new LocalStrategy(
  function(username, password, done) {
    User.findOne({ username: username }, function (err, user) {
      if (err) { return done(err); }
      if (!user) { return done(null, false); }
      if (!user.verifyPassword(password)) { return done(null, false);
      return done(null, user);
    });
  }
));
```

```
app.post('/login',
  passport.authenticate('local', { failureRedirect: '/login' }),
  function(req, res) {
    res.redirect('/');
  });
```

Passport JS

passport-jwt

Build Status Code Climate

A Passport strategy for authenticating with a JSON Web Token.

This module lets you authenticate endpoints using a JSON web token. It is intended to be used to secure RESTful endpoints without sessions.

Usage

An example configuration which reads the JWT from the http Authorization header with the scheme 'bearer':

```
var JwtStrategy = require('passport-jwt').Strategy,
    ExtractJwt = require('passport-jwt').ExtractJwt;
var opts = {}
opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
opts.secretOrKey = 'secret';
opts.issuer = 'accounts.examplesoft.com';
opts.audience = 'yoursite.net';
passport.use(new JwtStrategy(opts, function(jwt_payload, done) {
    User.findOne({id: jwt_payload.sub}, function(err, user) {
        if (err) {
            return done(err, false);
        }
        if (user) {
            return done(null, user);
        } else {
            return done(null, false);
            // or you could create a new account
        }
    });
}));
```

```
npm install passport-jwt
```

```
yarn add passport-jwt
```

```
var JwtStrategy = require('passport-jwt').Strategy,
    ExtractJwt = require('passport-jwt').ExtractJwt;
var opts = {}
opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
opts.secretOrKey = 'secret';
opts.issuer = 'accounts.examplesoft.com';
opts.audience = 'yoursite.net';
passport.use(new JwtStrategy(opts, function(jwt_payload, done) {
    User.findOne({id: jwt_payload.sub}, function(err, user) {
        if (err) {
            return done(err, false);
        }
        if (user) {
            return done(null, user);
        } else {
            return done(null, false);
            // or you could create a new account
        }
    });
}));
```

- O Passport suporta a persistência de dados de autenticação na sessão do usuário. Além disso fornece middleware para proteção de rotas. Isso permite que você restrinja o acesso a determinadas rotas apenas a usuários autenticados.

```
const requireJWTAuth = passport.authenticate("jwt", { session: false });

app.get("/clientes", requireJWTAuth, async (req, res) => {
    try {
        const clientes = await db.any("SELECT * FROM clientes;");
        console.log("Retornando todos clientes.");
        res.json(clientes).status(200);
    } catch (error) {
        console.log(error);
        res.sendStatus(400);
    }
});
```

Passport JS

```
passport.use(
    new LocalStrategy(
        {
            usernameField: "username",
            passwordField: "password",
        },
        async (username, password, done) => {
            try {
                // busca o usuário no banco de dados
                const user = await db.oneOrNone(
                    "SELECT * FROM users WHERE user_id = $1;",
                    [username],
                );

                // se não encontrou, retorna erro
                if (!user) {
                    return done(null, false, { message: "Usuário incorreto." });
                }

                // verifica se o hash da senha bate com a senha informada
                const passwordMatch = await bcrypt.compare(
                    password,
                    user.user_password,
                );

                // se senha está ok, retorna o objeto usuário
                if (passwordMatch) {
                    console.log("Usuário autenticado!");
                    return done(null, user);
                } else {
                    // senão, retorna um erro
                    return done(null, false, { message: "Senha incorreta." });
                }
            } catch (error) {
                return done(error);
            }
        },
    ),
);
```

Passport JS

```
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;

passport.use(
    new JwtStrategy(
        {
            jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
            secretOrKey: "your-secret-key",
        },
        async (payload, done) => {
            try {
                const user = await db.oneOrNone(
                    "SELECT * FROM users WHERE user_id = $1;",
                    [payload.username],
                );

                if (user) {
                    done(null, user);
                } else {
                    done(null, false);
                }
            } catch (error) {
                done(error, false);
            }
        },
    ),
);
```

JWT - JSON Web Token

- JSON Web Token (JWT) é um padrão aberto (RFC 7519) que define uma maneira compacta e autossuficiente de representar informações entre duas partes.

- Essas informações podem ser verificadas e confiáveis, e são tipicamente usadas para autenticação e autorização em aplicações web e serviços.

- Os middlewares jsonwebtoken e bcrypt são bibliotecas comuns usadas em aplicações Node.js para diferentes propósitos, especialmente relacionados à autenticação e segurança.

```
const jwt = require("jsonwebtoken");
const bcrypt = require("bcrypt");
```

- No exemplo, jwt.sign() é usado para criar um token JWT.
- A chave secreta é usada para assinar e verificar a autenticidade do token
- Outro comando comum é jwt.verify(), usado para verificar e decodificar um token JWT.

```
const token = jwt.sign({ username: req.body.username }, "your-secret-key", {
    expiresIn: "1h",
});
```

/login

- Utiliza a estratégia de autenticação local do Passport para verificar as credenciais do usuário.
- A opção { session: false } indica que a autenticação não deve ser baseada em sessão, pois o JWT será usado para autenticação stateless.
- A autenticação stateless significa que o servidor não mantém nenhum estado ou informação sobre o cliente entre as solicitações.

```
app.post("/login", passport.authenticate("local", { session: false }), (req, res) => {

    // Cria o token JWT
    const token = jwt.sign({ username: req.body.username }, "your-secret-key", {
        expiresIn: "1h",
    });

    res.json({ message: "Login successful", token: token });
});
```

- O callback é executado após a autenticação bem-sucedida. Dentro deste callback, um token JWT é gerado e enviado de volta ao cliente.

- Responde ao cliente com um objeto JSON contendo a mensagem "Login successful" e o token JWT gerado.

- Esse token pode ser usado pelo cliente para autenticação em requisições subsequentes.

```
app.post("/login", passport.authenticate("local", { session: false }), (req, res) => {

    // Cria o token JWT
    const token = jwt.sign({ username: req.body.username }, "your-secret-key", {
        expiresIn: "1h",
    });

    res.json({ message: "Login successful", token: token });
});
```