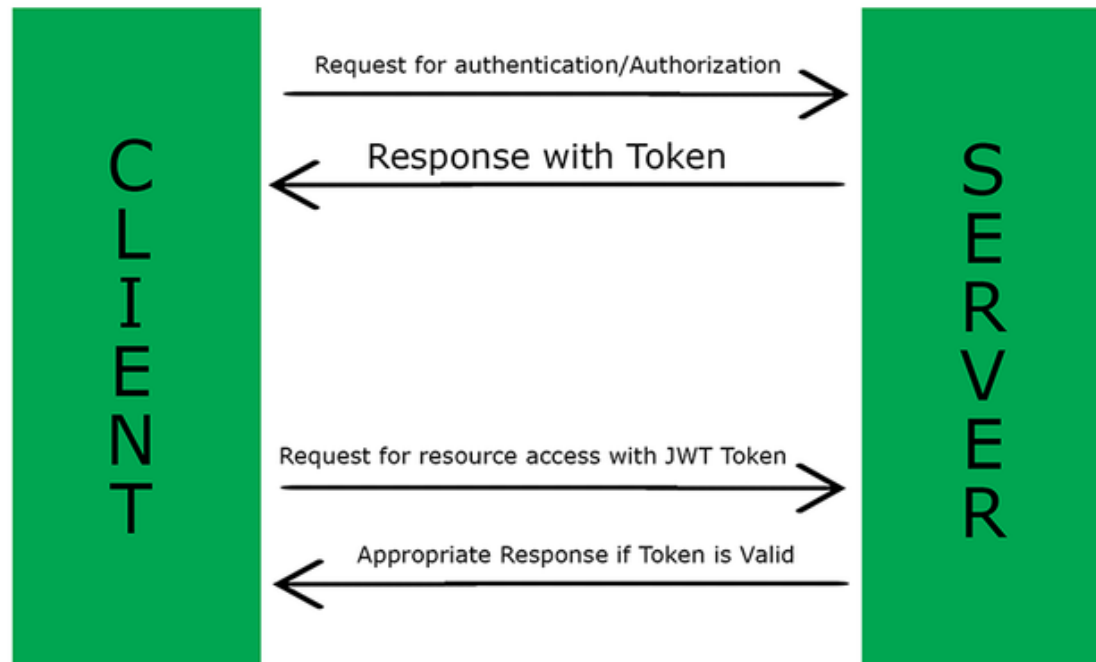


# Autentykacja w Express

# Autentykacja JWT



JWT - JSON Web Tokens

<https://www.geeksforgeeks.org/how-to-implement-jwt-authentication-in-express-js-app/>

# JSON Web Tokens

## JSON Web Tokens (JWT):

- jest otwartym standardem przemysłowym (RFC 7519) definiującym bezpieczne metody wymiany danych pomiędzy stronami z wykorzystaniem JSON; wykorzystywany często do autentykacji.

**JWT** jest kodowany przy użyciu standardu ***base64url*** i składa się z:

- nagłówek w formacie JSON zawierający algorytm użyty do zabezpieczenia tokena oraz typ tokena;
- dane (token i inne) w formacie JSON;
- sygnatura tokena pozwalająca na weryfikację jego autentyczności.

W aplikacji webowej Express można skorzystać z pakietu **jsonwebtoken**, który implementuje odpowiednie funkcje do obsługi JWT:

*npm install jsonwebtoken*

# Pakiet *bcrypt*

W aplikacji webowej, która korzysta z bazy danych do przechowywania danych autentykacyjnych zaleca się przechowywanie ich w bezpiecznej postaci, tj. w postaci zahaszowanej.

## **bcrypt:**

- jest algorytmem przeznaczonym do **hashowania** haseł opartym o algorytm szyfrowania **Blowfish**
- jego konstrukcja zdecydowanie utrudnia ataki typu **bruteforce**; wykorzystuje tzw. **salt** (sól - dodatkowa losowa liczba dodawana przed obliczeniem hasha) co zapobiega atakom z wykorzystaniem "*Rainbow Tables*".

# Pakiet *bcrypt*

W celu wykorzystania *bcrypt* w aplikacji można zainstalować pakiet o tej samej nazwie:

```
npm install bcrypt
```

Poniżej instrukcja wykorzystania - wywołanie asynchroniczne (z dokumentacji):

```
// To hash a password:
```

```
//Technique 1 (generate a salt and hash on separate function calls):
```

```
bcrypt.genSalt(saltRounds, function(err, salt) {  
  bcrypt.hash(myPlaintextPassword, salt, function(err, hash) {  
    // Store hash in your password DB.  
  });  
});
```

```
//Technique 2 (auto-gen a salt and hash):
```

```
bcrypt.hash(myPlaintextPassword, saltRounds, function(err, hash) {  
  // Store hash in your password DB.  
});
```

# Pakiet *bcrypt*

Parametry wywołania :

- **saltRounds** (cost factor) - im większa liczba rund tym więcej czasu algorytm potrzebuje na wyliczenie hash'a, ale jednocześnie hash jest bardziej odporny na ataki 'brute force'; w systemach produkcyjnych saltRounds nie powinno mieć wartości mniejszej niż 12
- **salt** - liczba losowa, generowana automatycznie, powinna być różna dla różnych obliczeń tak, że nawet dla tych samych haseł hash będzie inny; salt jest przechowywany w bazie razem z zahaszowanym stringiem w czytelnej postaci.
- **function**(err, hash) - funkcja callback, która otrzymuje w parametrze *err* i *hash*, a której ciało należy wypełnić samemu; można (a nawet jest to bardziej preferowane) też zrezygnować z funkcji callback i przy pomocy *await* 'odebrać wynik' (przykład w pomocy).

**ZADANIE**

# Zadanie 1 - autentykacja

Dodaj do aplikacji "Gallery" system autentykacji tak, aby:

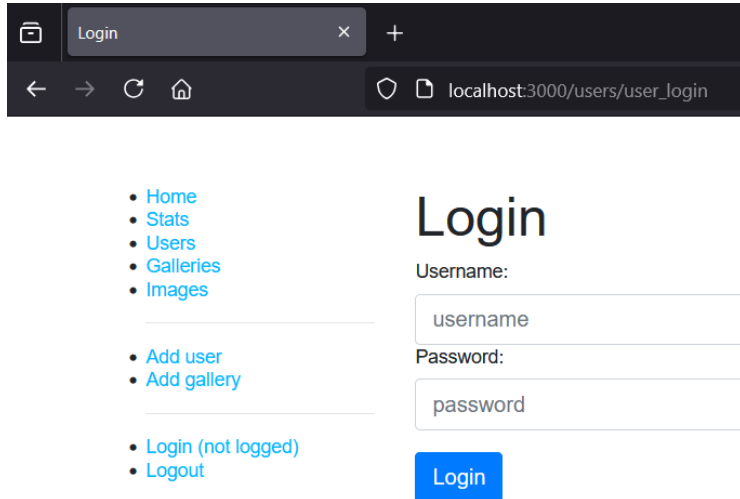
- a) użytkownik mógł się zalogować i wylogować; po stronie klienta *token* powinien być przechowywany w pliku *cookie*,
- b) tylko użytkownik zalogowany może dodawać nową galerię, a dodana galeria będzie automatycznie przypisana do niego;
- c) wyjątek to użytkownik o loginie "admin" - ma on prawo do dodawania galerii w imieniu dowolnego użytkownika.

W zadaniu konieczne będzie ponowne zmodyfikowanie modelu *User* (dodanie pola: *password*) i zmodyfikowanie formularza dodawania użytkownika (dodanie input Password).



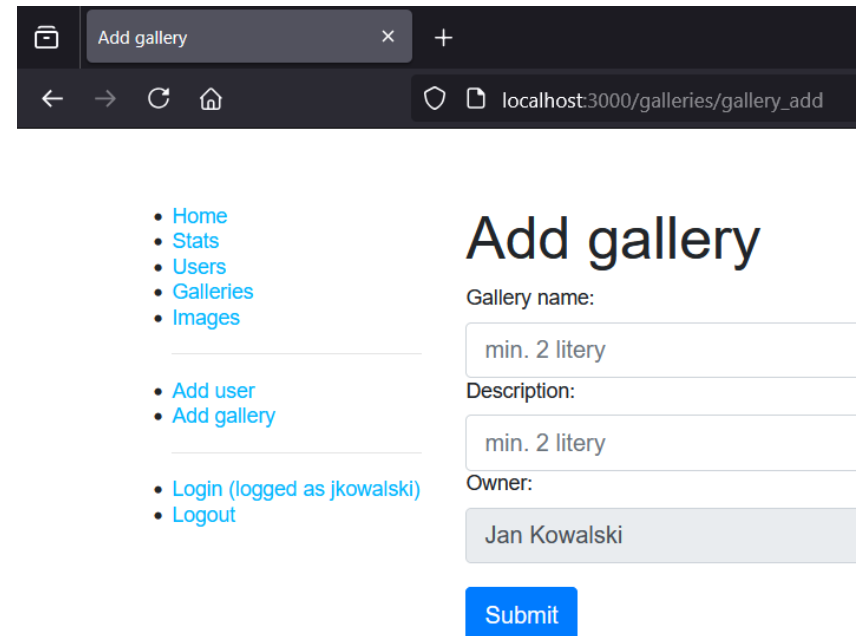
# Zadanie 1

Strona logowania:



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/users/user\_login'. The page has a dark header with a 'Login' tab. On the left, there is a sidebar menu with links: Home, Stats, Users, Galleries, Images, Add user, Add gallery, Login (not logged), and Logout. The main content area is titled 'Login' and contains two input fields: 'Username:' with the placeholder 'username' and 'Password:' with the placeholder 'password'. A blue 'Login' button is positioned below the password field.

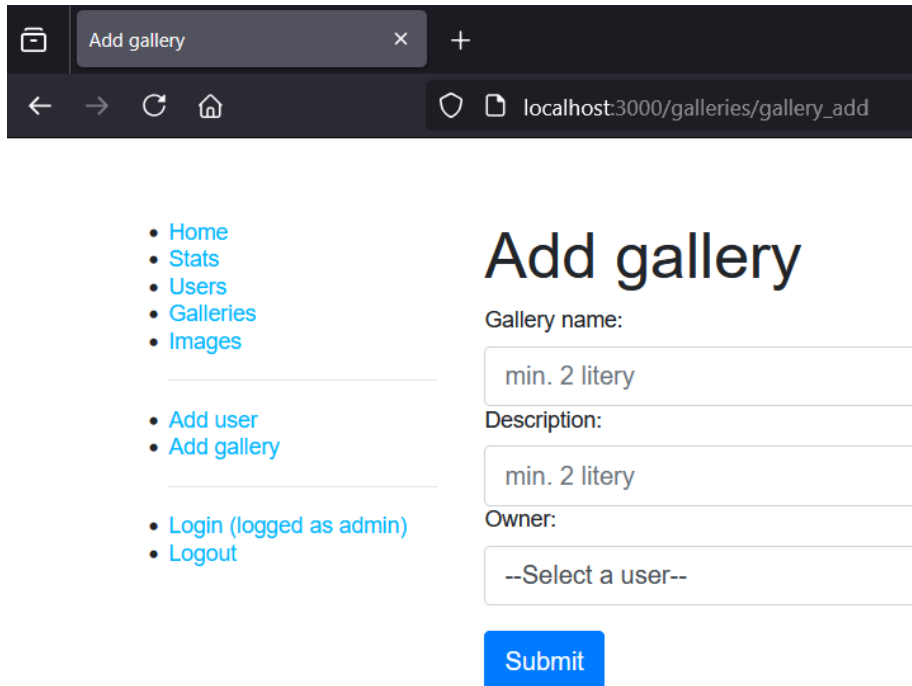
Strona dodawania galerii przez zwykłego użytkownika (lista rozwijana jest nieaktywna, można też w ogóle z niej zrezygnować):



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/galleries/gallery\_add'. The page has a dark header with an 'Add gallery' tab. On the left, there is a sidebar menu with links: Home, Stats, Users, Galleries, Images, Add user, Add gallery, Login (logged as jkowalski), and Logout. The main content area is titled 'Add gallery' and contains three input fields: 'Gallery name:' with the placeholder 'min. 2 litery', 'Description:' with the placeholder 'min. 2 litery', and 'Owner:' with the value 'Jan Kowalski'. A blue 'Submit' button is positioned below the owner field.

# Zadanie 1

Strona dodawania galerii przez *admina*  
(lista rozwijana jest aktywna):



• Home

• Stats

• Users

• Galleries

• Images

• Add user

• Add gallery

• Login (logged as admin)

• Logout

## Add gallery

Gallery name:

min. 2 litery

Description:

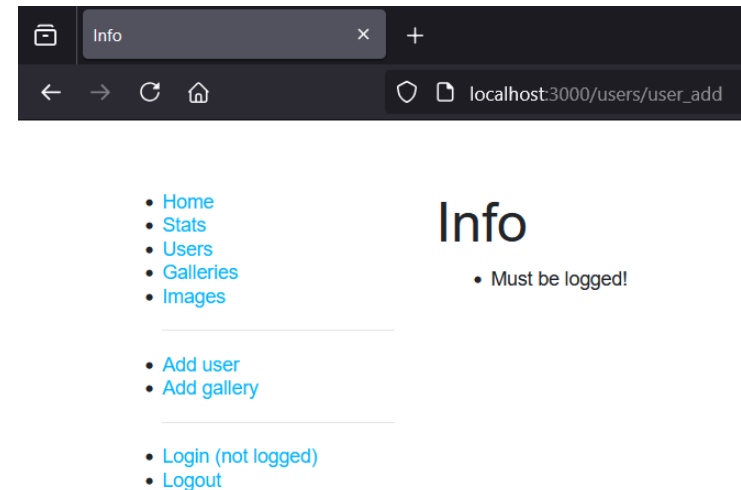
min. 2 litery

Owner:

--Select a user--

Submit

Komunikat w przypadku próby wejścia na stronę dodawania galerii bez uprzedniego zalogowania (tymczasowo może być zwykły res.send):



• Home

• Stats

• Users

• Galleries

• Images

• Add user

• Add gallery

• Login (not logged)

• Logout

## Info

- Must be logged!

# Zadanie 1a - pomoc

- 1) Doinstalować pakiety z *jsonwebtoken* i *bcrypt*.
- 2) Do modelu użytkownika (**./models/user.js**) dodać nowe pole *password*.

```
const mongoose = require("mongoose");

const Schema = mongoose.Schema;

const UserSchema = new Schema({
  name: { type: String, maxLength: 100 },
  surname: { type: String, maxLength: 100 },
  username: { type: String, maxLength: 100 },
  password: { type: String }
}, { collection: 'users' });

// Export model
module.exports = mongoose.model("User", UserSchema);
```

# Zadanie 1a - pomoc

3) Dodać do widoku formularza dodawania użytkownika (`./views/user_form.pug`) nowy *input* password do pobierania hasła

```
label(for='password') Password:  
    input#password.form-control(type='password', minlength="8" placeholder='min. 8  
    litery' name='password' autocomplete='off' required value='') )
```

# Zadanie 1a - pomoc

4) Dodać do kontrolera obsługi danych formularza dodawania użytkownika (POST) (**./controllers/userController.js**) walidację i obsługę hasła.

```
...  
body("password", "Password to short!")  
  .isLength({ min: 8 } ),  
...  
  
//zaszyfrowanie hasła - bcrypt.hash działa asynchronicznie  
const passwordHash = await bcrypt.hash(req.body.password, 10)  
  
// Tworzenie obiektu/dokumentu newuser z modelu User po 'oczyszczeniu' danych  
const newuser = new user({  
  name: req.body.name,  
  surname: req.body.surname,  
  username: req.body.username,  
  password: passwordHash,  
});  
...  
...
```

# Zadanie 1a - pomoc

5) W **./routes/users.js** dodać nowe ścieżki GET i POST do obsługi formularzy logowania oraz wylogowania.

```
// USER LOGIN GET
router.get("/user_login", userController.user_login_get);
// USER LOGIN POST
router.post("/user_login", userController.user_login_post);

// USER LOGOUT GET
router.get("/user_logout", userController.user_logout_get);
```

# Zadanie 1a - pomoc

6) W `./controllers/userController.js` zaimportować wymagane moduły i dodać funkcję wyświetlania formularza logowania (GET).

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

// GET - Kontroler wyświetlania formularza logowania
exports.user_login_get = (req, res, next) => {
  res.render("user_login_form", { title: "Login" });
}
```

# Zadanie 1a - pomoc

7) W `./controllers/userController.js` należy dodać funkcję obsługi danych z formularza logowania (POST). W przypadku powodzenia logowania, zostanie wygenerowany token i zapisany w cookie na komputerze klienta.

```
// POST - Kontroler przetwarzania formularza logowania
exports.user_login_post = (req, res, next) => {
  let username = req.body.username;
  let password = req.body.password;

  user.findOne({ username })
    .then((user) => {
      if (user) {
        bcrypt.compare(password, user.password, function (err, result) {
          if (err) {
            res.render("user_login_form", { title: "Login", messages: ["Some bcrypt errors!"] });
            return;
          }
          if (result) {
            let token = jwt.sign({ username: user.username }, 'kodSzyfrujacy', { expiresIn: '1h' });
            res.cookie('mytoken', token, { maxAge: 600000 });
            // Login OK
            res.render('index', { title: 'Express', loggedInUser: user.username });
          } else {
            // Bad pass
            res.render("user_login_form", { title: "Login", messages: ["Bad pass!"] });
          }
        })
      }
    })
  } else {
    // No user found!
    res.render("user_login_form", { title: "Login", messages: ["No user found!"] });
  }
}
```



# Zadanie 1a - pomoc

8) W `./controllers/userController.js` należy dodać funkcję obsługi wylogowania się (GET).

```
// GET - Kontroler wylogowania
exports.user_logout_get = (req, res, next) => {
  res.clearCookie('mytoken')

  res.render('index', { title: 'Express' });
}
```

# Zadanie 1a - pomoc

9) W szablonie bazowym **./views/layout.pug** dodać linki do ścieżek logowania i wylogowania (poniżej fragment kodu). Można też dodać informację o zalogowaniu się (loggedUser), ale wtedy trzeba w jakiś sposób przekazywać tę informację do szablonów przy renderowaniu odpowiedzi.

```
li
  hr
  if loggedUser
    a(href='/users/user_login') Login (logged as #{loggedUser})
  else
    a(href='/users/user_login') Login (not logged)
li
  a(href='/users/user_logout') Logout
```

# Zadanie 1a - pomoc

10) Należy utworzyć formularz logowania `./view/user_login_form.pug`, poniżej przykład.

```
extends layout

block content
  h1=title

  form(method='POST')
    div.form-group
      label(for='username') Username:
      input#username.form-control(type='text', placeholder='username' name='username' required value='')
      label(for='password') Password:
      input#password.form-control(type='password', placeholder='password' name='password' autocomplete='off'
required value='')
      button.btn.btn-primary(type='submit') Login

  if messages
    ul
      for message in messages
        li!= message
```

*Uwaga!*

*Aby przetestować działanie autentykacji najlepiej z bazy usunąć "starych" użytkowników, ich galerie, a potem dodać ich ponownie tym razem za pomocą formularza z hasłem.*

# Zadanie 1b - pomoc

1) Należy także utworzyć moduł `./middleware/authenticate.js` np. z funkcją `authenticate` i ją wyeksportować. Będzie ona wykorzystywana do sprawdzania tokena sesji przesyłanego od klienta.

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

const authenticate = (req, res, next) => {
  try {
    // przysłany token
    const token = req.cookies.mytoken;
    // dekoduj token
    const decode = jwt.verify(token, 'kodSzyfrujacy');
    // dodanie do request (req.user) danych zweryfikowanego usera
    req.user = decode;
    req.loggedUser = req.user.username;
    next();
  }
  catch (err) {
    res.redirect('/');
    // res.render("info", { title: "Info", messages: ['Must be logged!'] });
  }
}

module.exports = authenticate
```

# Zadanie 1b - pomoc

2) W `./routes/galleries.js` importujemy `./middleware/authenticate` i dodajemy wywołanie funkcji `authenticate` przed przekazaniem requestu dalej do funkcji kontrolera dodawania galerii (poniżej fragment kodu).

```
// Middleware autentykacji
const authenticate = require('../middleware/authenticate');

// ...

// GALLERY ADD GET (/galleries/gallery_add)
router.get("/gallery_add", authenticate, gallery_controller.gallery_add_get);

// GALLERY ADD POST (/galleries/gallery_add)
router.post("/gallery_add", authenticate, gallery_controller.gallery_add_post);
```

# Zadanie 1c - pomoc

W **./controllers/galleryController.js** należy zmodyfikować funkcję kontrolera dodawania galerii - trzeba weryfikować *username* zalogowanego użytkownika i w zależności od tego renderować np. inny formularz (najlepiej w utworzyć nowy widok np. **./views/gallery\_form\_user**, przeznaczony dla zwykłego użytkownika)

Aby przetestować działanie autentykacji najlepiej usunąć "starych" użytkowników i dodać ich ponownie tym razem z formularzem z hasłem