

NodeJS

API Authentication and Security

Розробити систему авторизації

Завдання 1. При реєстрації та при оновленні користувача пароль зберігати в зашифрованому вигляді;

Завдання 2. Створити запит `/users/login` на вхід


Завдання 3. Примінити авторизацію: в запиті відправити `authToken` в заголовку `Authorization`. Сервер отримує токен, верифікує його і авторизує користувача (або ні в іншому випадку).

Завдання 4. Створити запити `/users/logout`, `/users/logoutAll`. Вимагає авторизації. Видаляє запис про токен із БД.

Завдання 1. При реєстрації та при оновленні користувача пароль потрібно зберігати в зашифрованому вигляді

Крок 1.1. Схема даних і модель

Переконайтесь в наявності схеми userSchema та моделі даних User:

```
let userSchema = new mongoose.Schema({  
  name: {type: String...},  
  password: {type: String...},  
  age: {type: Number...},  
  email: {type: String...},  
});
```

78

```
const User = mongoose.model('User', userSchema);
```

Крок 1.2. Шифрування паролю

Інсталюйте і підключіть модуль bcrypt в моделі user. Викличте для схеми метод pre(), що спрацюватиме щоразу перед викликом

```
73 // Перед збереженням хешує пароль
74 userSchema.pre('save', async function(next) {
75     //Отримуємо екземпляр даного користувача
76     const user = this;
77     //Якщо модифікується пароль
78     if (user.isModified('password')) {
79         //Зашифруємо його
80         user.password = await bcrypt.hash(user.password, 8);
81     }
82     next();
83 });
```

Крок 1.3. Створіть обробку на редагування даних (PATCH)

В обробнику для редагування даних в маршрутизаторі *user* замість методу `findByIdAndUpdate()` рекомендується реалізувати `router.patch("/users/:id", async (req, res) => {`

```
    try {
      const user = await User.findOne({_id: req.params.id});
      if (!user) {
        res.status(404);
        throw new Error("User not found");
      }
      const fields = ["firstName", "lastName", "age", "password"];
      fields.forEach((field) => {
        if (req.body[field]) {
          user[field] = req.body[field];
        }
      })
      await user.save();
      res.json(user);
    } catch (error) {
      res.send(error.message);
    }
  })
```

Крок 1.4. Переконайтесь в шифруванні пароля при відправці запитів на реєстрацію та на редагування

TaskApp / User / RegisterUser

POST {{url}}/users

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "firstName": "John",
3   "lastName": "Smith",
4   "age": 20,
5   "password": "123456",
6   "email": "john@gmail.com"
7 }
```

Body Cookies Headers (7) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   "email": "john@gmail.com",
3   "password": "$2b$08$4nYtmw.Icm2JQwbPMXg36.z1ntL1uR8wQQ2UaqUn4j24K4XQSG7Im",
4   "firstName": "John",
5   "lastName": "Smith",
6   "age": 20,
7   "_id": "641ac80c1c5243cd6eeb25da",
8   "tokens": [],
9   "__v": 0
10 }
```

TaskApp / User / UpdateUser

PATCH {{url}}/users/641ac80c1c5243cd6eeb25da

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "password": "654321",
3   "firstName": "Johnny"
4 }
```

Body Cookies Headers (7) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

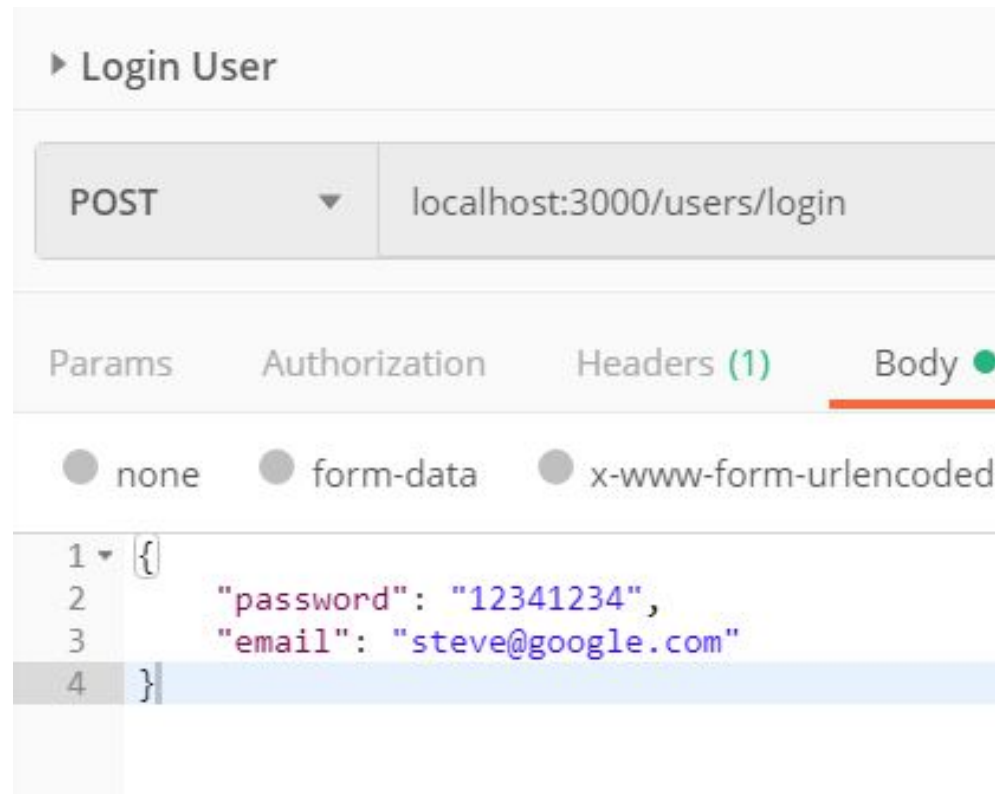
```
1 {
2   "_id": "641ac80c1c5243cd6eeb25da",
3   "email": "john@gmail.com",
4   "password": "$2b$08$.wTxo10uzV2ci2Ac0ReLxuCA0YEZulnnfSh0vBakGkFGIwPyo2Gs6",
5   "firstName": "Johnny",
6   "lastName": "Smith",
7   "age": 20,
8 }
```

Завдання 2. Створити запит `/users/login` на вхід.

- Здійснити пошук користувача по email та верифікацію паролю.
- Якщо успішна автентифікація, потрібно залогінитись: згенерувати token, зберегти його в БД, а також відправити в клієнт.
- Клієнт повинен зберегти token в змінній оточення `authToken`.

Крок 2.1. Створіть запит /users/login

1) В Postman створіть та збережіть запит POST /users/login



Крок 2.2. Створіть метод для аутентифікації

Для моделі даних реалізуйте статичний метод `findOneByCredentials()`, який перевірятиме правильність авторизації

```
58  userSchema.statics.findOneByCredentials = async (email, password) => {
59      const user = await User.findOne({email});
60
61      if (!user) {
62          throw new Error('Incorrect email');
63      }
64
65      const isMatch = await bcrypt.compare(password, user.password);
66      if (!isMatch) {
67          throw new Error('Incorrect password');
68      }
69      return user;
70  };
```

Крок 2.3. Викличте метод аутентифікації

В обробнику `/users/login` аутентифікуйте користувача:

```
19 router.post("/users/login", async (req, res) => {  
20     try {  
21         const user = await User.findOneByCredentials(req.body.email, req.body.password);  
22         res.send(user);  
23     } catch (e) {  
24         res.status(400).send()  
25     }  
26 });
```

Крок 2.4. Протестуйте аутентифікацію

Перевірте виконання запиту при вірно і невірно введених даних

POST `{{url}}/users/login`

Params Authorization Headers (9) **Body** Pre-request Sc

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ bir

```
1 {
2   "password": "asdfasdf",
3   "email": "steve@google.com"
4 }
```

Body Cookies Headers (6) Test Results

Pretty Raw Preview JSON

```
1 {
2   "user": {
3     "age": 21,
4     "_id": "5cbecfb4327a183e3092b898",
5     "name": "Maria",
6     "password": "$2b$08$fZWbPFm7tsV0wAhfilrmpuf8VbH0jt:",
7     "email": "steve@google.com",
8     ".": "7"
9   }
10 }
```

Login User

POST `{{url}}/users/login`

Params Authorization Headers (9)

☐ none ☐ form-data ☐ x-www-form

```
1 {
2   "password": "12341234",
3   "email": "steve@google.com"
4 }
```

Body Cookies Headers (6) Test Results

Pretty Raw Preview HTML

```
1 Incorrect password
```

POST `{{url}}/users/login`

Params Authorization Headers (9)

☐ none ☐ form-data ☐ x-www-form-u

```
1 {
2   "password": "asdfasdf",
3   "email": "steve1@google.com"
4 }
```

Body Cookies Headers (6) Test Results

Pretty Raw Preview HTML

```
1 Incorrect email
```

Крок 2.5. Далі залогінімо користувача. Для цього спочатку в схемі даних потрібно створити масив tokens

В схемі даних моделі створимо нову властивість tokens – масив токенів

```
6 let userSchema = new mongoose.Schema({
7   name: {type: String...},
12  password: {type: String...},
17  age: {type: Number...},
26  email: {type: String...},
37  tokens: [{
38    token: {
39      type: String,
40      required: true
41    }
42  }]
43 });
```

Крок 2.6. Створимо метод для генерації токена

Встановіть jsonwebtoken і підключіть його в змінній jwt
Для схеми даних моделі створимо метод generateAuthToken:

```
45 userSchema.methods.generateAuthToken = async function () {  
46     //Для зручності отримаємо екземпляр користувача  
47     const user = this;  
48     //Генеруємо токен  
49     const token = jwt.sign({_id: user._id.toString()}, 'kdweueksdsjfi');  
50     //Приєднуємо токен в масив tokens даного користувача  
51     user.tokens = user.tokens.concat({token});  
52     //Зберігаємо в БД  
53     await user.save();  
54     //Повертаємо токен  
55     return token;  
56 }
```


Крок 2.7. Забезпечимо генерацію токену при успішній аутентифікації

Після виклику `findOneByCredentials()`, викличемо `user.generateAuthToken()`

```
20 router.post("/users/login", async (req, res) => {
21   try {
22     const user = await User.findOneByCredentials(req.body.email, req.body.password);
23     const token = await user.generateAuthToken();
24     res.send({user, token});
25   } catch (e) {
26     res.status(400).send()
27   }
28 });
```

Крок 2.8. Тестуємо

Переконайтеся у створенні токенів при запиті логування /users/login:



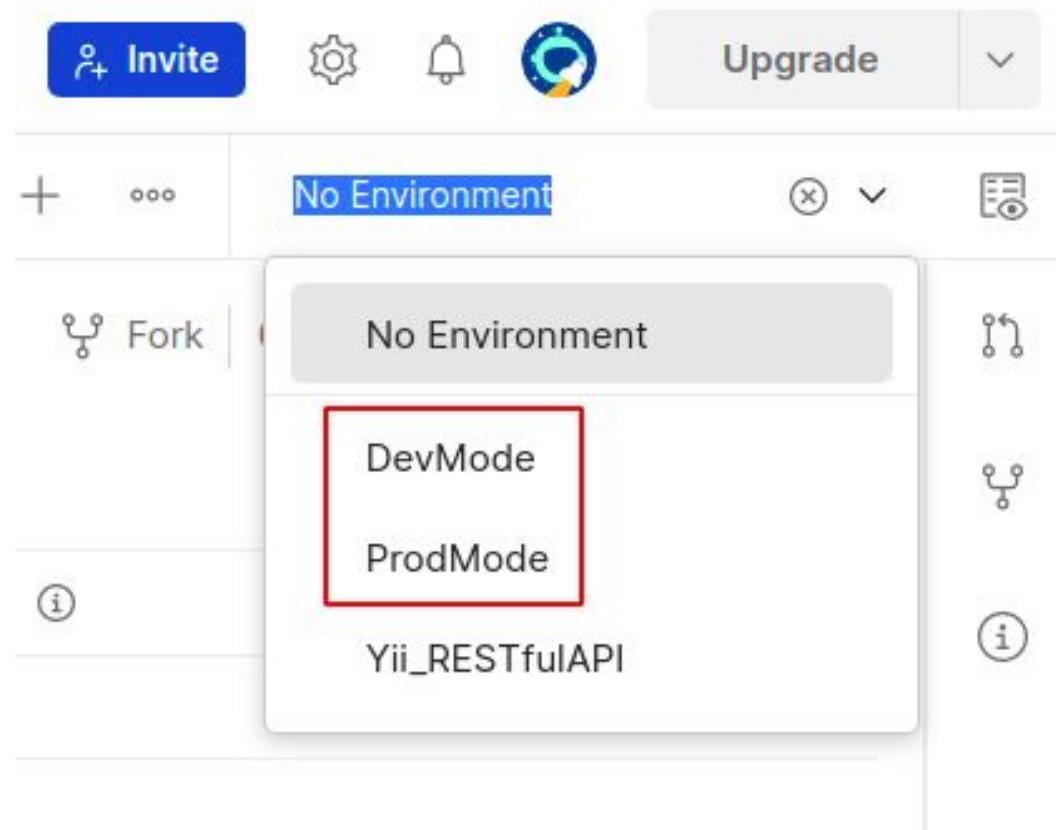
```
POST {{url}}/users/login

Pretty Raw Preview JSON ↵

1 {
2   "user": {
3     "age": 21,
4     "_id": "5cbecfb4327a183e3092b898",
5     "name": "Maria",
6     "password": "$2b$08$fZWbPFm7tsV0wAhfilrmpuf8VbH0jtXfIJ0FntkvFVFjLBhn5RPs2",
7     "email": "steve@google.com",
8     "__v": 8,
9     "tokens": [
10    {
11      "_id": "5cbee9a4b64a6c38dcb19538",
12      "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI1Y2JlY2ZiNDMyN2ExODNlMzA5MmI4OTgiLCJpYXQiOiJlNTYwMTU1MjR9.6kOKy1S2hd-Du_sNIz6FhZXnvp98kw8iGxE9zGXqYA0"
13    },
```


Крок 2.9. Середовище розробки і робоче середовище

- В Postman створіть два середовища DevMode і ProdMode



Крок 2.10. Змінна оточення *url*

- В середовищі DevMode створіть змінну *url* із значенням localhost:3000
- Використайте змінну *url* у всіх існуючих запитах

Створення змінної

DevMode	
Variable	Initial value
url	http://localhost:3000

Використання змінної

TaskApp / User / GetUsers

GET ▼ {{url}}/users

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Type Inherit auth... ▼

Крок 2.11

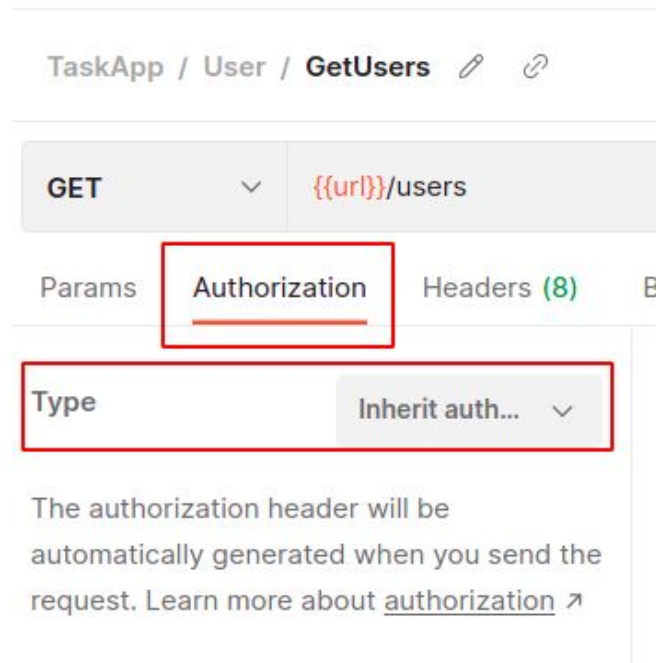
- 1) В Postman на рівні папки для наших запитів створити метод авторизації Bearer Token і в полі Token задати змінну оточення `{{authToken}}`

The screenshot shows the Postman interface for the 'Task App' collection. The 'Authorization' tab is selected, indicated by a green dot and an orange underline. The 'Name' field is 'Task App'. Below the tabs, a message states: 'This authorization method will be used for every request in this collection. You can override this by specifying one in the request.' The 'TYPE' dropdown is set to 'Bearer Token'. A warning box on the right says: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)'. The 'Token' field contains the variable `{{authToken}}`. A link [Learn more about authorization](#) is also present.

- Таким чином всі запити у яких метод авторизації наслідуються від батьківського (крім логування та реєстрації користувачів) будуть використовувати токен авторизації

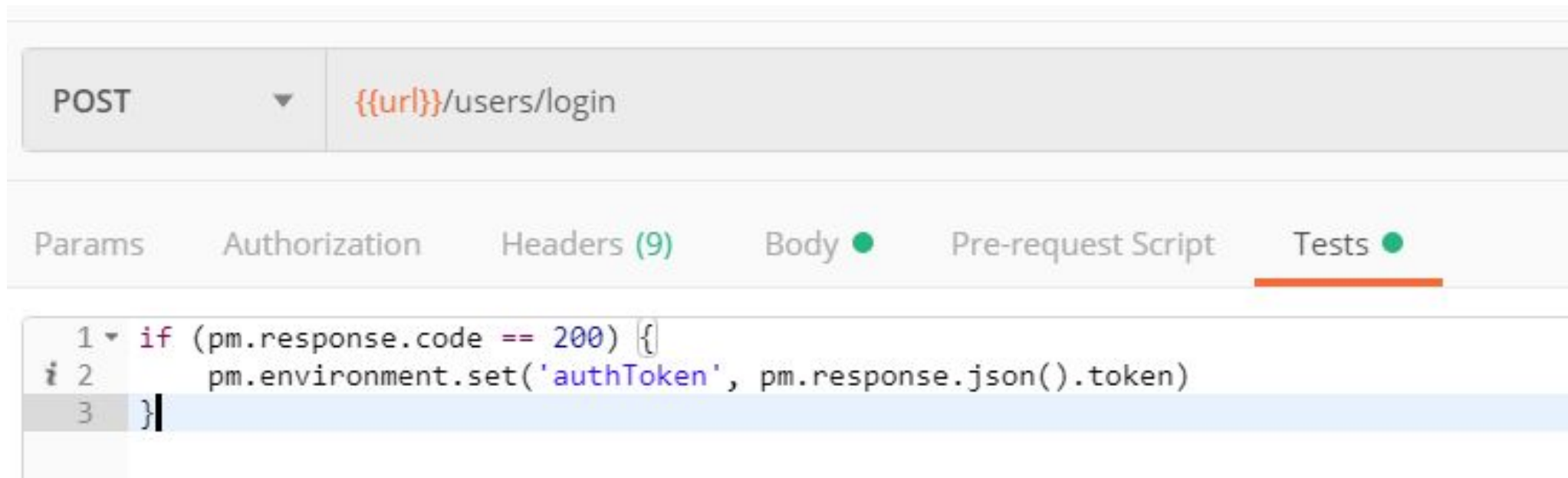
Крок 2.12

- Унаслідуйте метод авторизації для всіх запитів крім реєстрації користувача та авторизації користувача: inherit auth from parent



Крок 2.13. Автоматизуємо передачу значення змінній authToken

- У вкладці Tests для запиту users/login встановити нове значення для змінної середовища



Завдання 3. Примінити авторизацію:

- В запиті відправити authToken в заголовку Authorization
- Сервер отримує токен, верифікує його і авторизує користувача (або ні в іншому випадку)

Крок 3.1

- Створіть файл auth.js за шляхом src/middleware/auth.js з функцією для отримання токена, що надсилається в заголовці запиту
- Функція знаходить користувача за id, який вона отримала після декодування токена і записує його в req.user

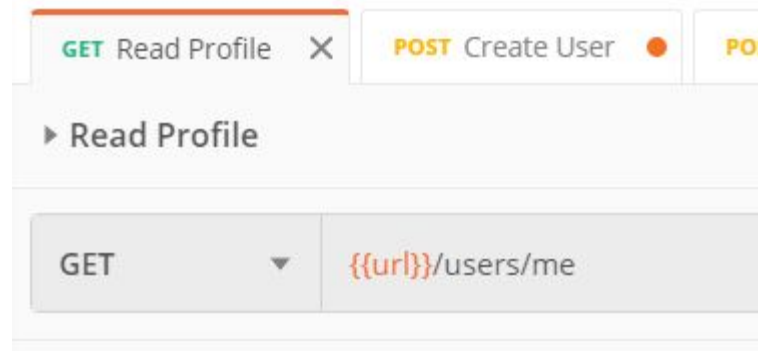
```
const jwt = require("jsonwebtoken");
const User = require("../models/user");

const auth = async (req, res, next) => {
  try {
    const token = req.header('Authorization').replace("Bearer ", "");
    const decoded = jwt.verify(token, 'kdweueksdsjfi');
    const user = await User.findOne({_id: decoded._id, 'tokens.token': token});
    if (!user) {
      throw new Error()
    }
    req.user = user;
    req.token = token;
    next();
  } catch (e) {
    res.status(401).send({error: "Please authenticate"});
  }
}

module.exports = auth;
```

Крок 3.2

- Створіть запит для перегляду даних про авторизованого користувача



Крок 3.3

- Здійснить обробку даного запиту з попереднім виконанням middleware-функції auth

```
52 router.get('/users/me', auth, async(req, res) => {  
53     res.send(req.user);  
54 })
```

Завдання 4. Створити запити
/users/logout, /users/logoutAll. Вимагає
авторизації. Видаляє запис про токен із
БД.

Крок 4.1

- Створити POST-запит users/logout

► Logout User

POST	{{url}}/users/logout
------	----------------------

```
31 router.post("/users/logout", auth, async(req, res) => {
32   try {
33     req.user.tokens = req.user.tokens.filter((token) => {
34       return token.token !== req.token;
35     })
36     await req.user.save()
37     res.send()
38   } catch (e) {
39     res.status(500).send()
40   }
41 })
```

Протестуйте виконання роботи

- Запити userLogin, userRegiser не потребують авторизації
- Всі інші запити потребують авторизації.

Послідовність тестування:

- Здійснюємо реєстрацію, вхід.
- Тестуємо запити
- Здійснюємо вихід
- Тестуємо запити