

Лабораторна робота №4

API Authentication and Security

Мета: ознайомитись з 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 заголовок 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...},
});

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

					ДУ «Житомирська політехніка».24.121.13.000 - Лр4			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Клосович І.А.			Звіт з лабораторної роботи		Лім.	Арк.
Перевір.		Сидорчук В.О.						Аркушів
Керівник							1	25
Н. контр.							ФІКТ Гр. ІПЗ-22-1[1]	
Зав. каф.								

```

4   const userSchema = new mongoose.Schema({
5       name: {type: String...},
10      password: {type: String...},
22      age: {type: Number...},
31      email: {type: String...}
44  })
45  const User = mongoose.model("User", userSchema)

```

Рис. 1. Перевірка наявності userSchema та моделі даних User

Крок 1.2. Шифрування паролю

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

```

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  });

```

```

new *
userSchema.pre( method: 'save', fn: async function (next :... ) {
    const user = this;
    if (user.isModified( path: 'password')) {
        user.password = await bcrypt.hash(user.password, salt: 8);
    }
    next();
});

```

Рис. 2. Результат

Крок 1.3. Створіть обробку на редагування даних (PATCH)

В обробнику для редагування даних в маршрутизаторі `user` замість методу `findByIdAndUpdate()` рекомендується реалізувати `save()`.

Приклад:

```
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);
  }
})
```

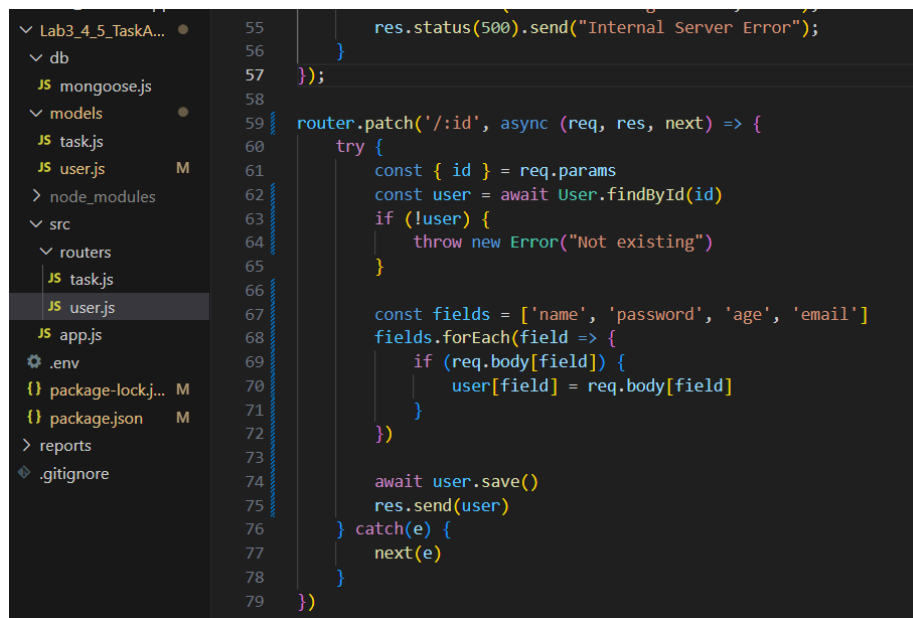
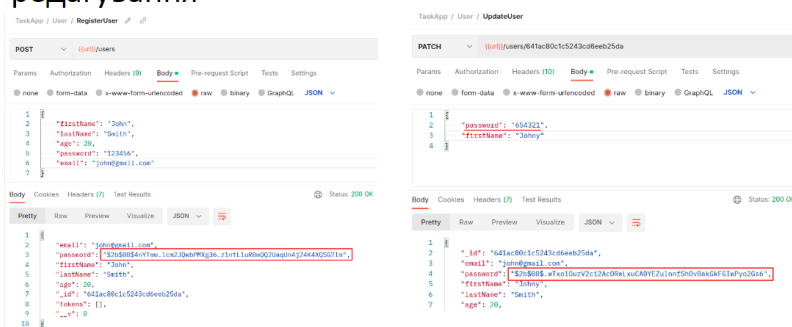


Рис. 3. Результат

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



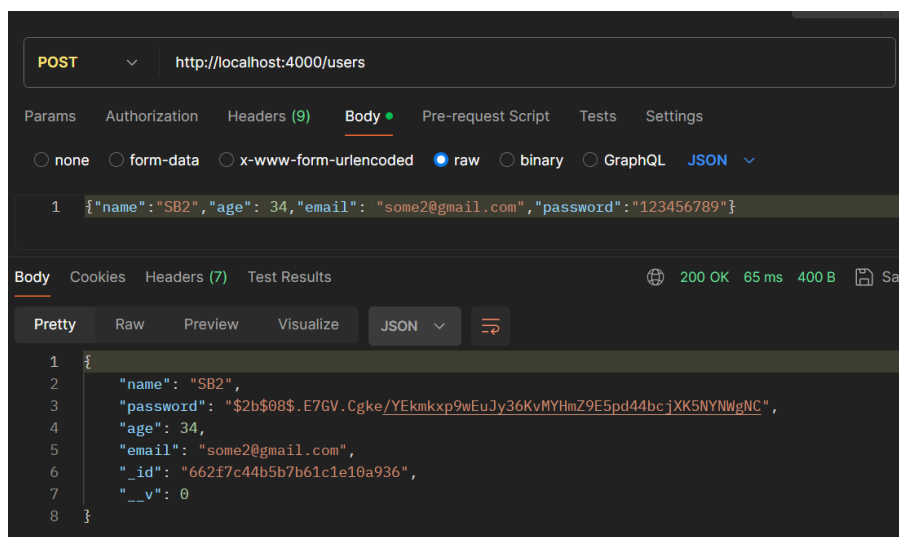


Рис. 4. Результат шифрування пароля під час реєстрації

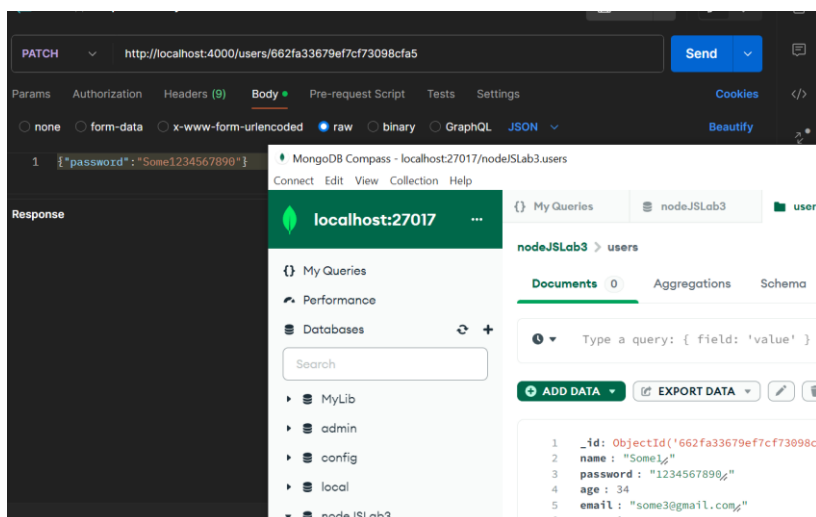


Рис. 5. До редагування

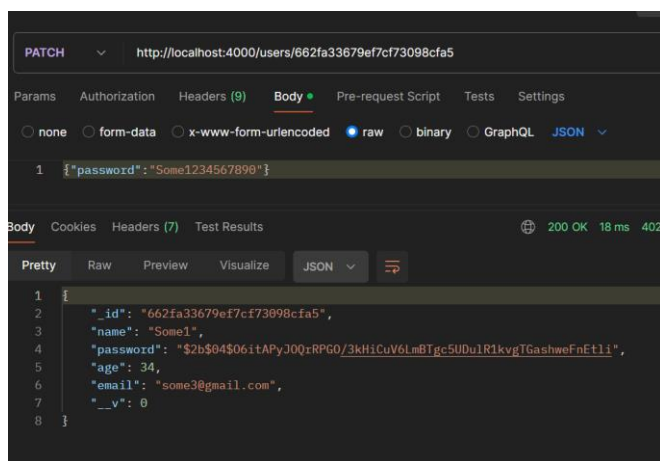


Рис. 6. Результат після оновлення

		Клосович І.А.			ДУ «Житомирська політехніка».24.121.13.000 - Лр4	Арк.
		Сидорчук В.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

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

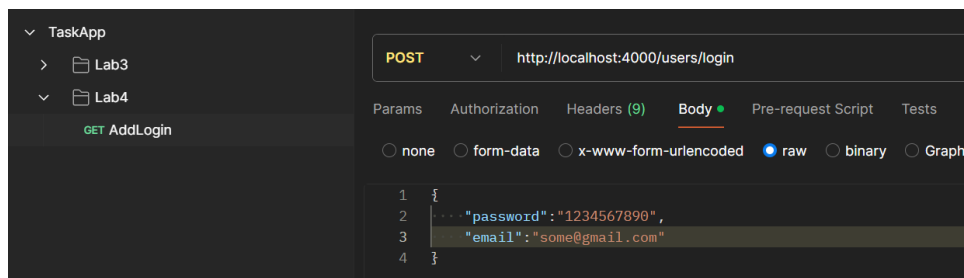
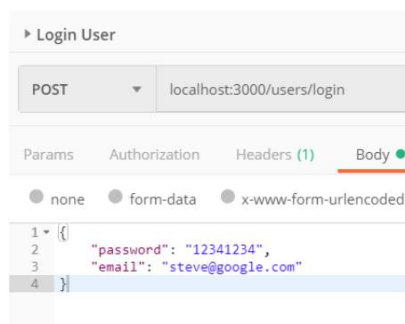


Рис. 7. Результат

		Клосович І.А.			ДУ «Житомирська політехніка».24.121.13.000 - Лр4	Арк.
		Сидорчук В.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		5

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 };
```

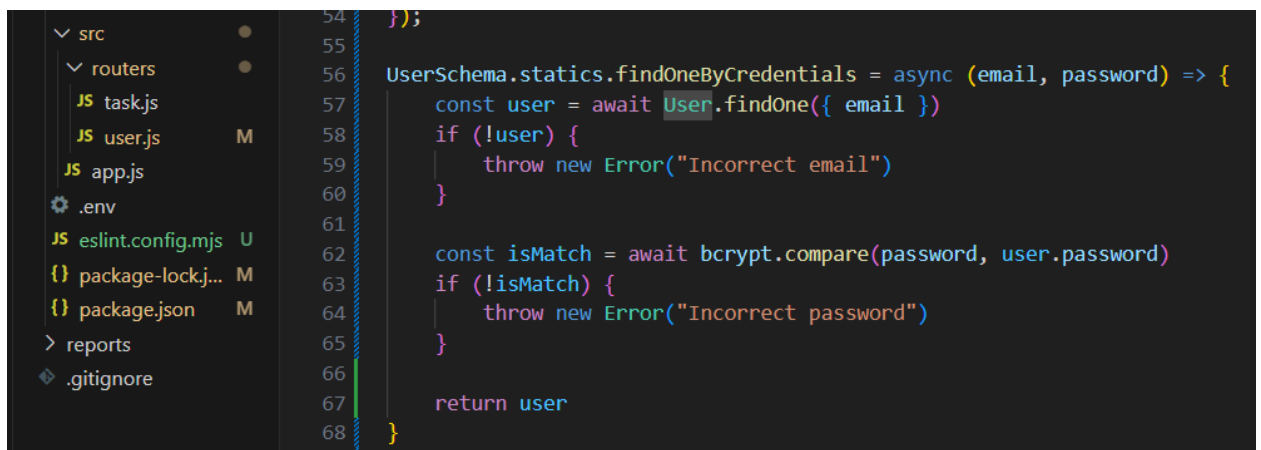


Рис. 8. Результат

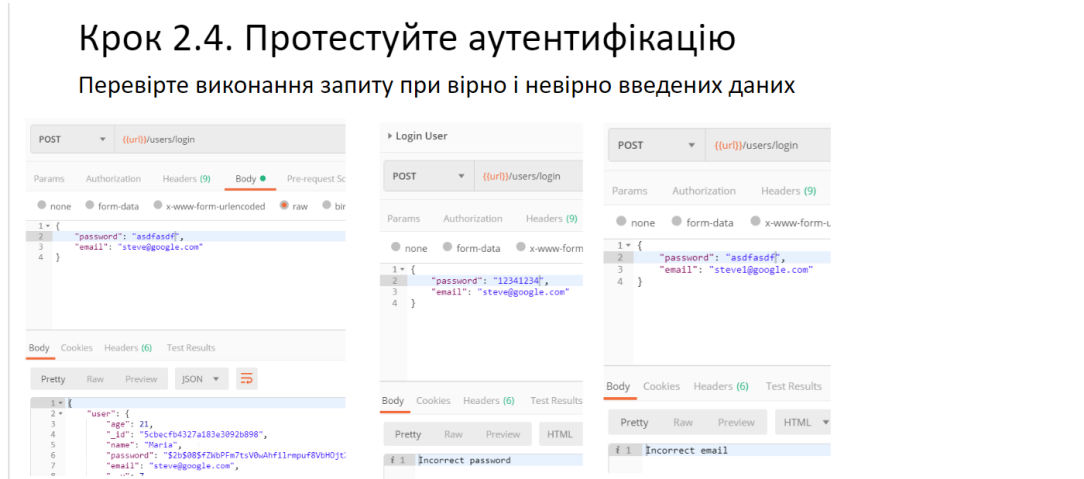
Крок 2.3. Викличте метод аутентифікації

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 });
```

```
80
81 router.post(['users/login', async (req, res) => {
82   try {
83     const user = await User.findOneByCredentials(req.body.email, req.body.password)
84     res.send(user)
85   } catch(e) {
86     res.status(400).send()
87   }
88 })
89
```

Рис. 9. Результат



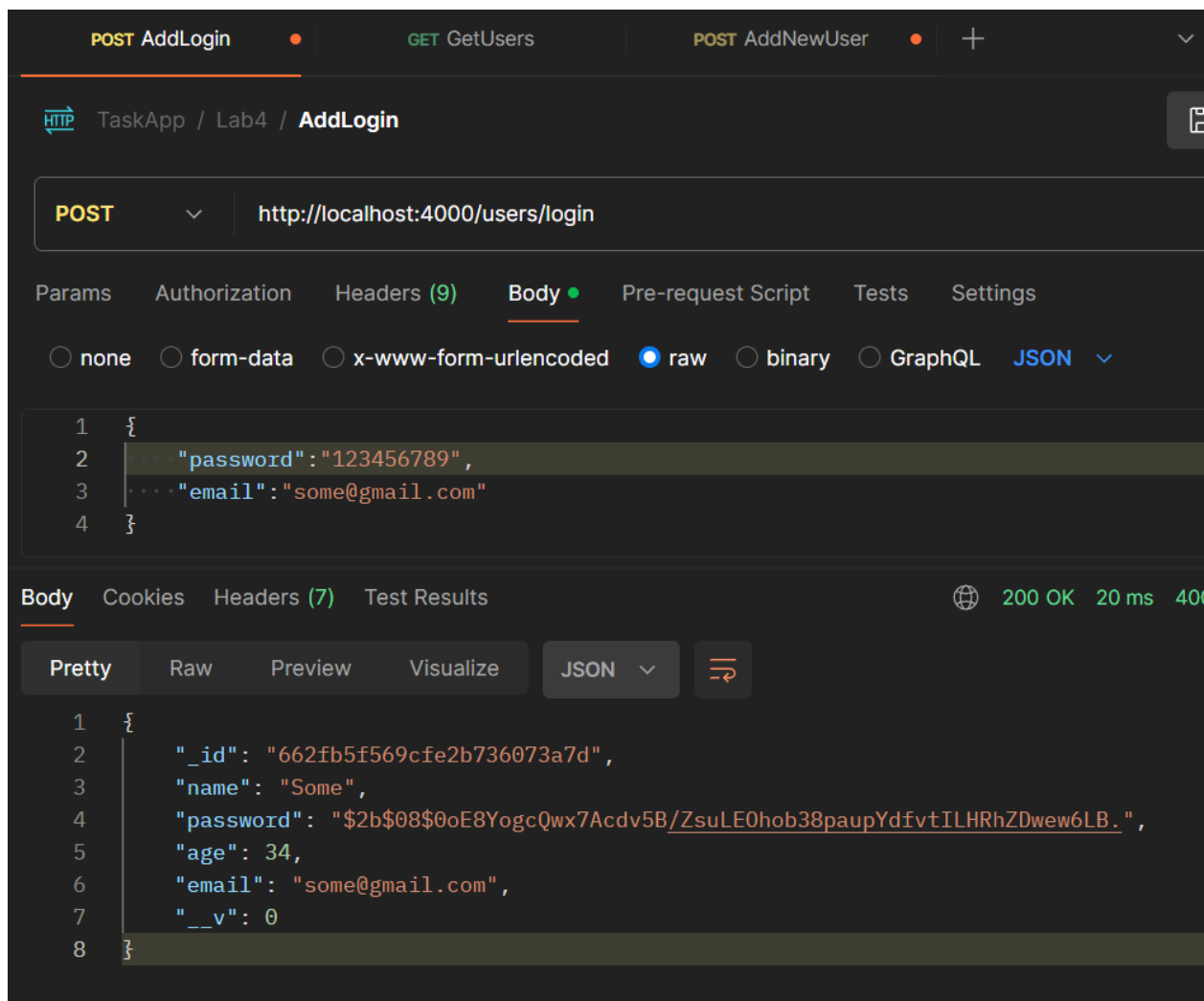


Рис. 10. Результат при коректних даних

		Клосович І.А.			ДУ «Житомирська політехніка».24.121.13.000 - Лр4	Арк.
		Сидорчук В.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

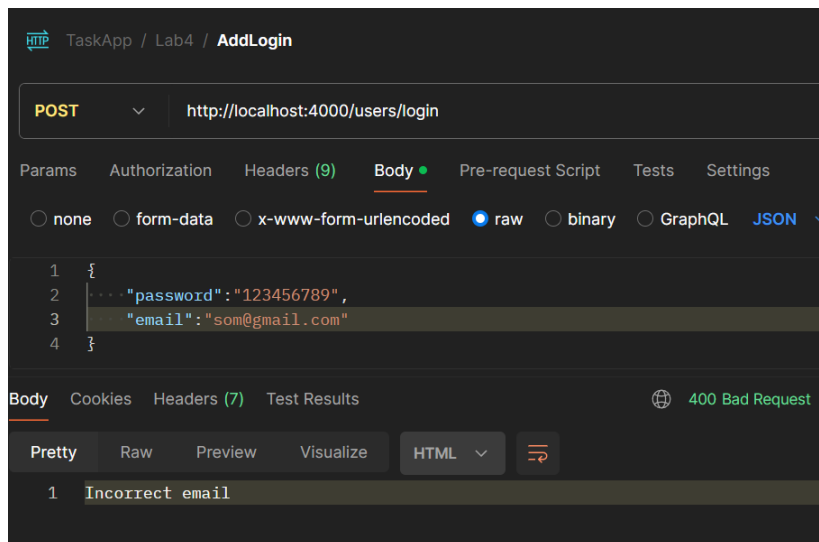


Рис. 11. Неправильний email

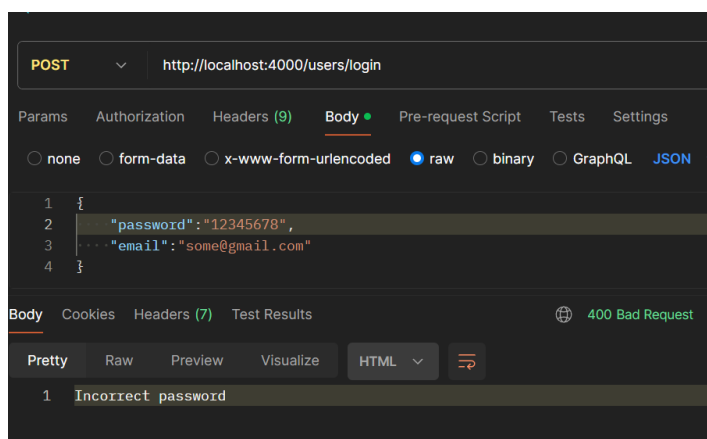


Рис. 12. Неправильний пароль

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 });

```

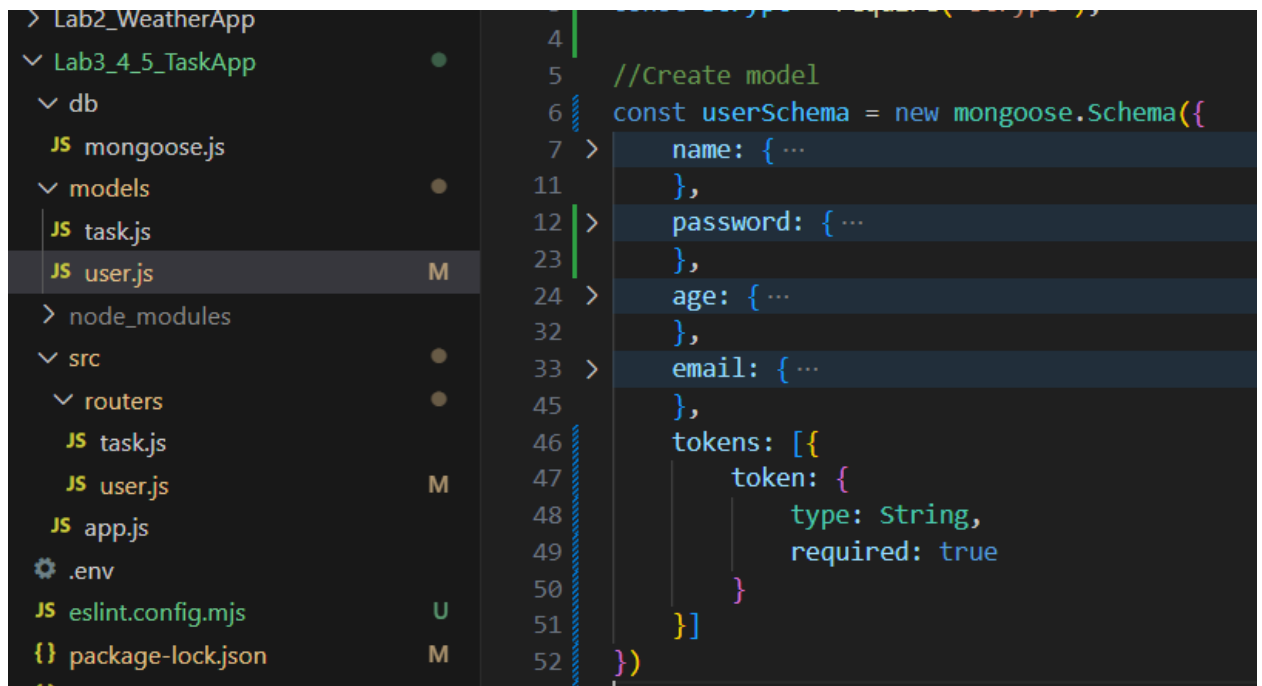


Рис. 13. Результат

Крок 2.6. Створимо метод для генерації токена

Встановить jsonwebtoken і підключить 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 };

```

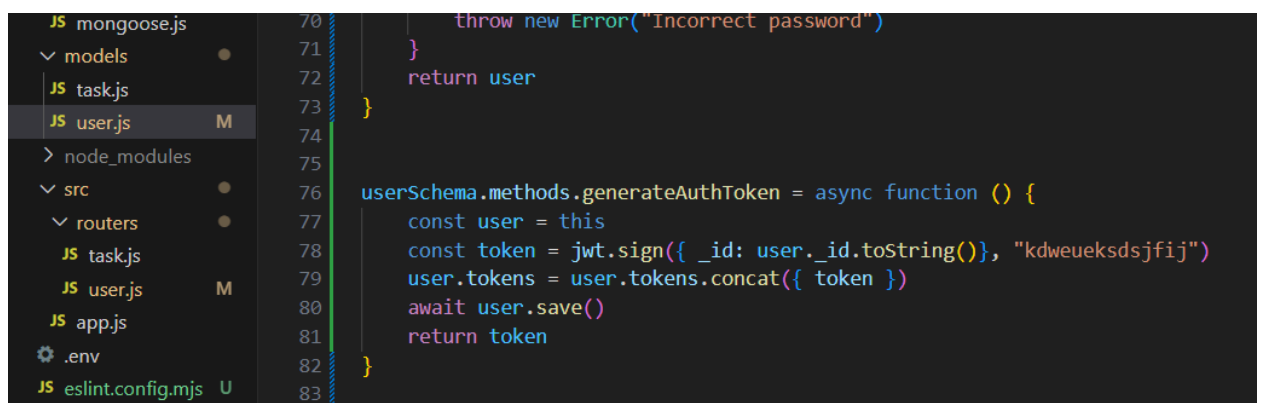


Рис. 14. Результат

		Клосович І.А.			ДУ «Житомирська політехніка».24.121.13.000 - Лр4	Арк.
		Сидорчук В.О.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

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 });
```

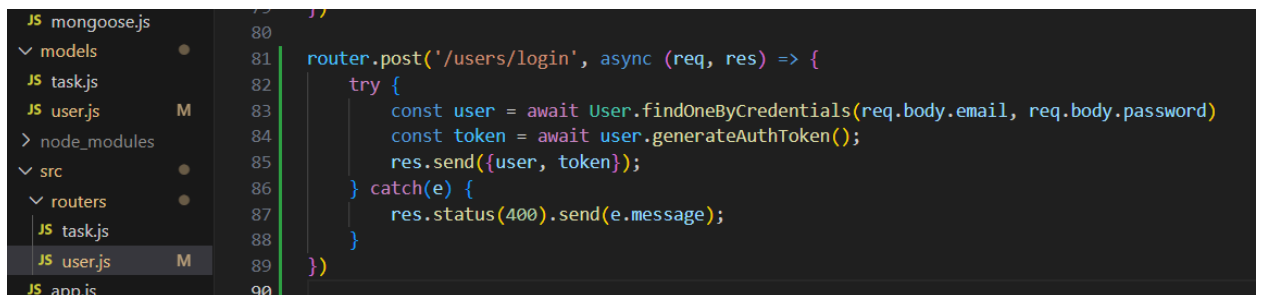


Рис. 15. Результат

Крок 2.8. Тестуємо

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



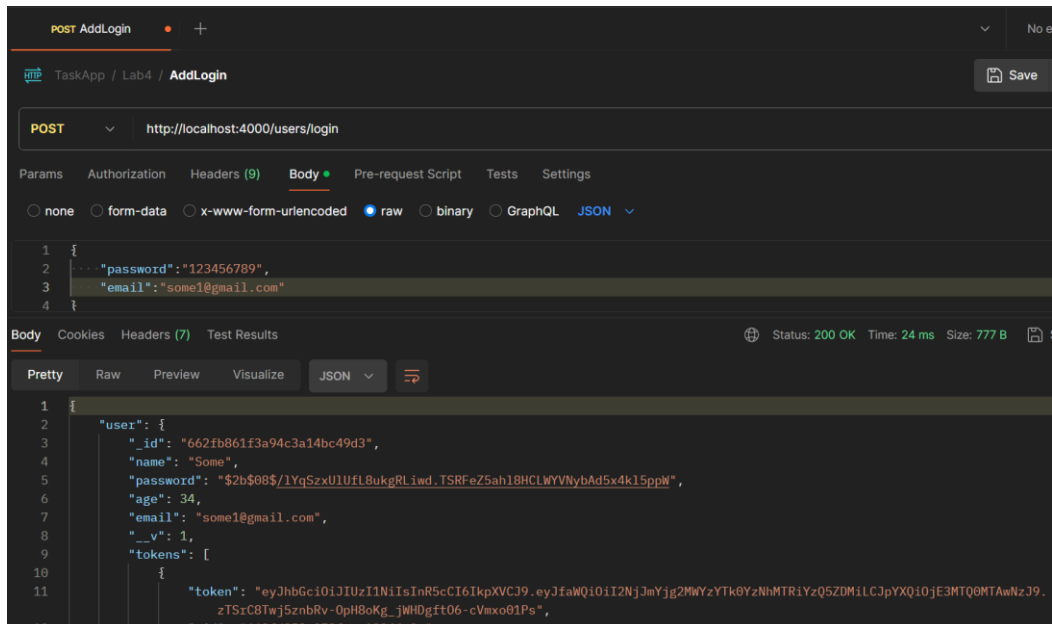
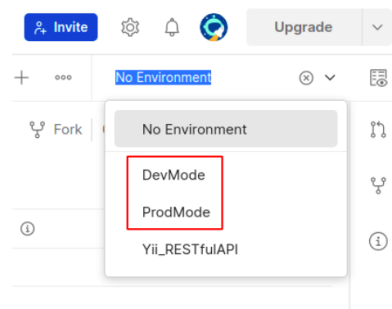


Рис. 16. Результат

Крок 2.9. Середовище розробки і робоче середовище

- В Postman створіть два середовища DevMode і ProdMode



		Клосович І.А.			ДУ «Житомирська політехніка».24.121.13.000 - Лр4	Арк.
		Сидорчук В.О.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

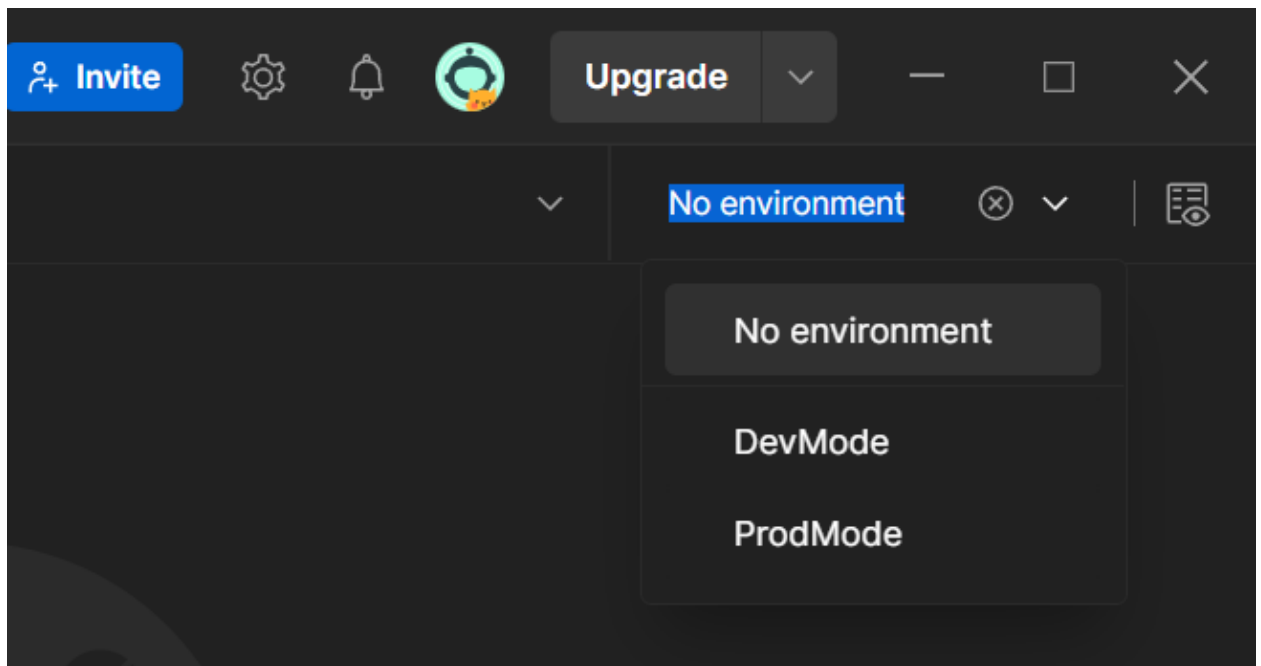


Рис. 17. Результат

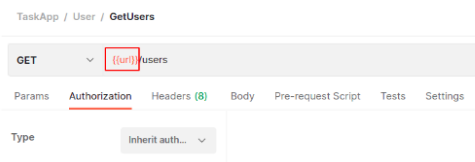
Крок 2.10. Змінна оточення *url*

- 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



DevMode		DevMode	
Filter variables		Fork	0
		Save	Share
Variable	Type	Initial value	Current value
<input checked="" type="checkbox"/> url	default	http://localhost:4000/	http://localhost:4000/
Add new variable			

Рис. 18. Результат

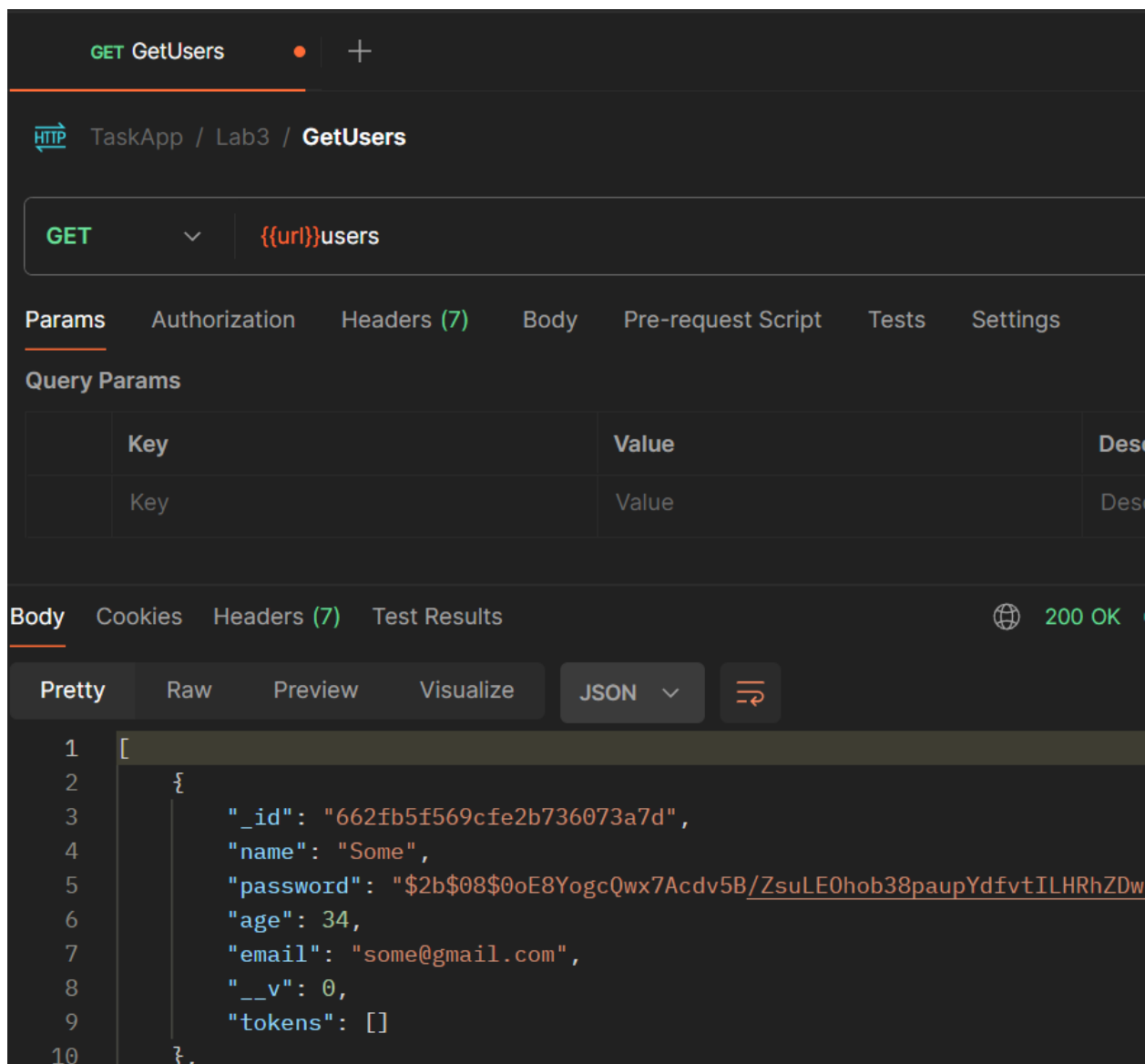
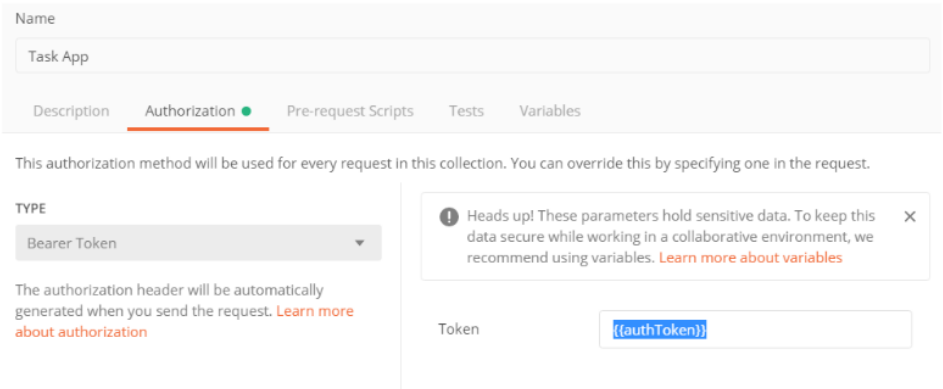


Рис. 19. Результат

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}}



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

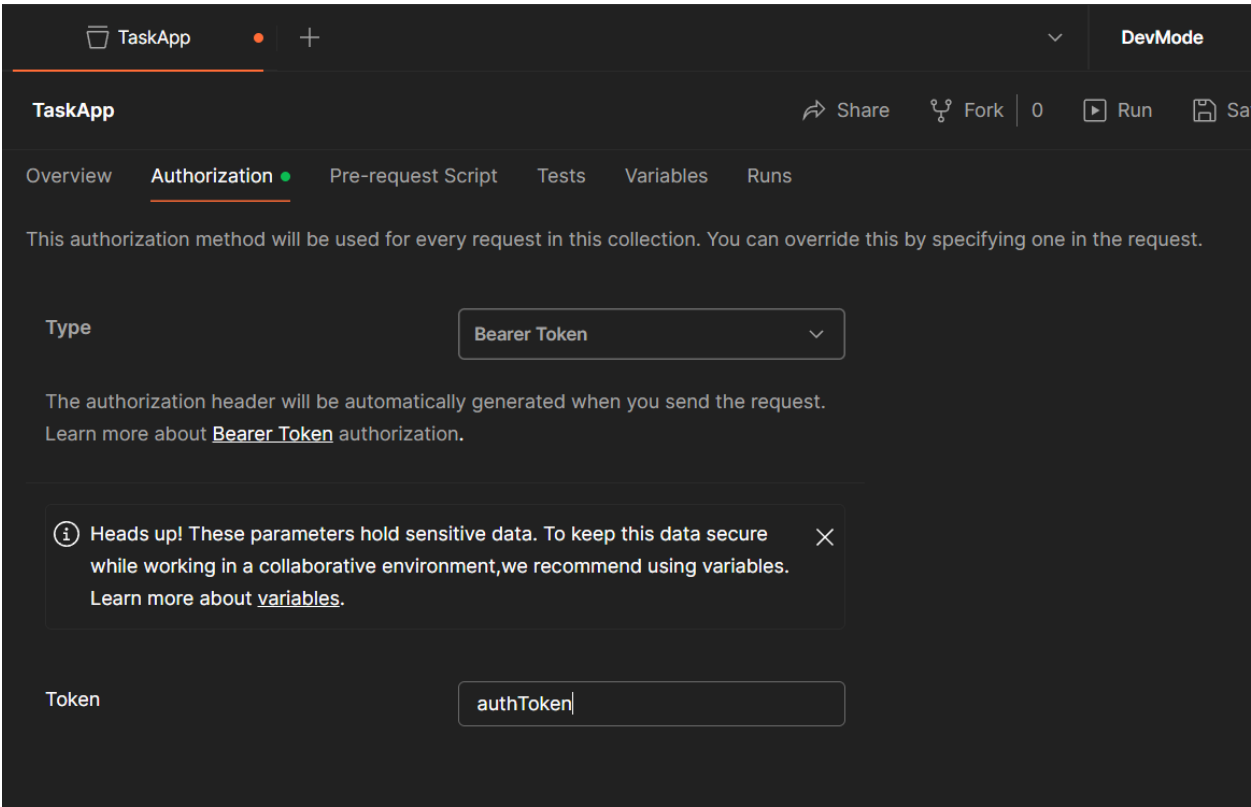


Рис. 20. Результат

Крок 2.12

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

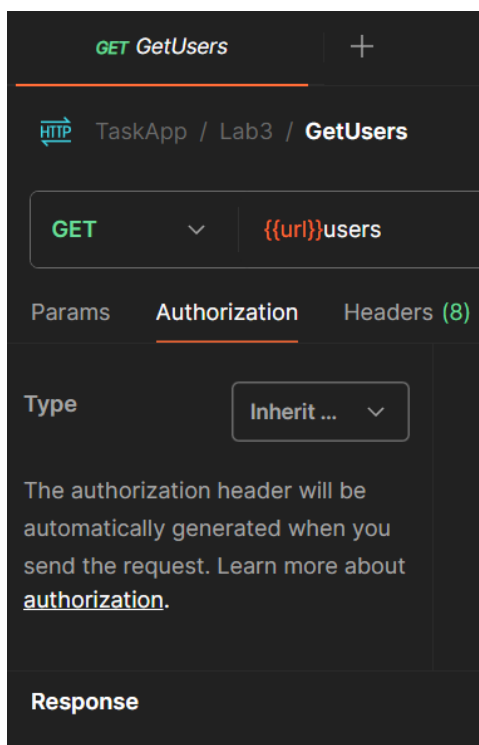
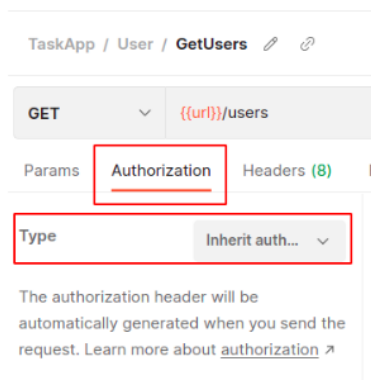


Рис. 21. Результат

		Клосович І.А.			ДУ «Житомирська політехніка».24.121.13.000 - Лр4	Арк.
		Сидорчук В.О.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

Крок 2.13. Автоматизуємо передачу значення змінній authToken

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

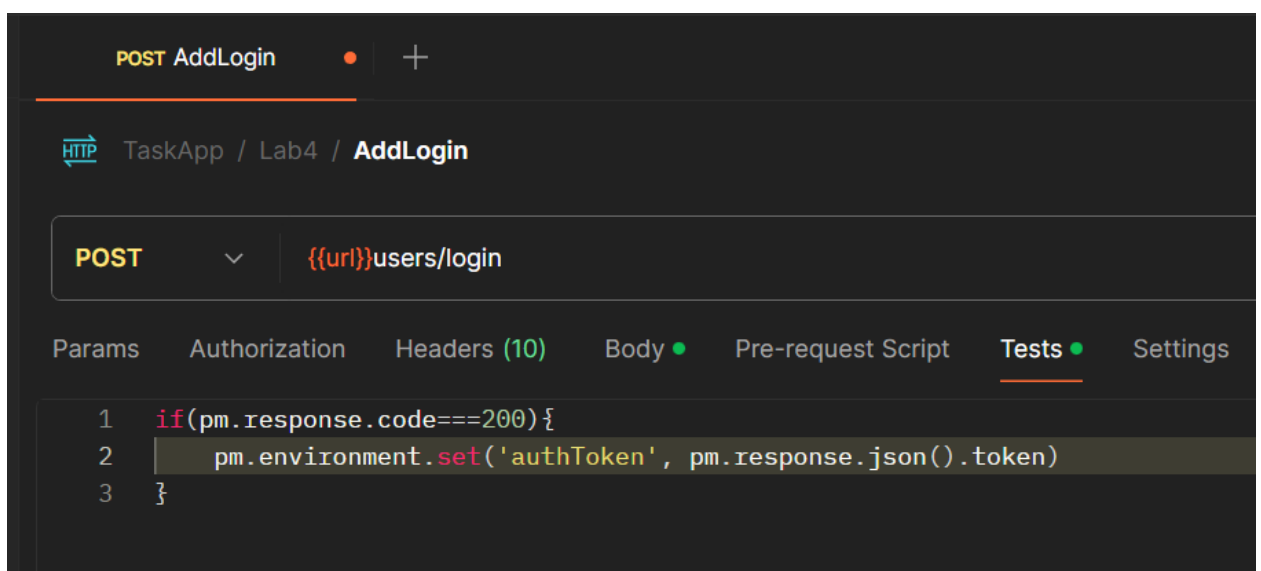
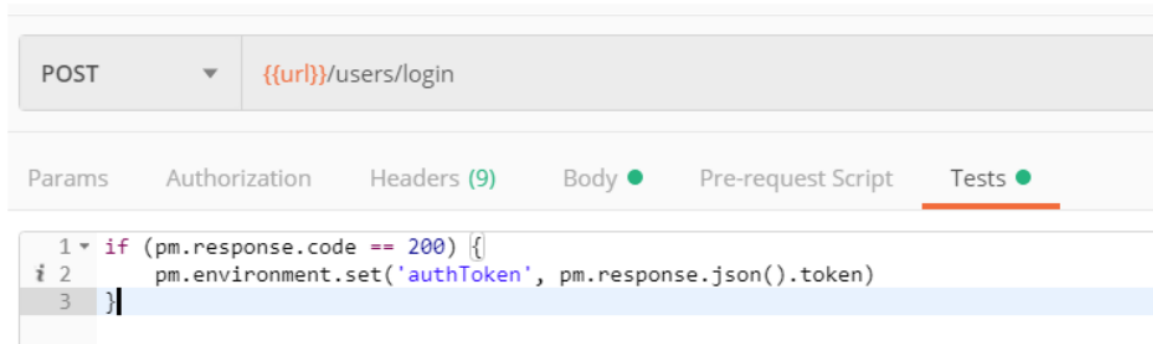


Рис. 22. Результат

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, 'kdweueksdsjfiij');
    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;
```

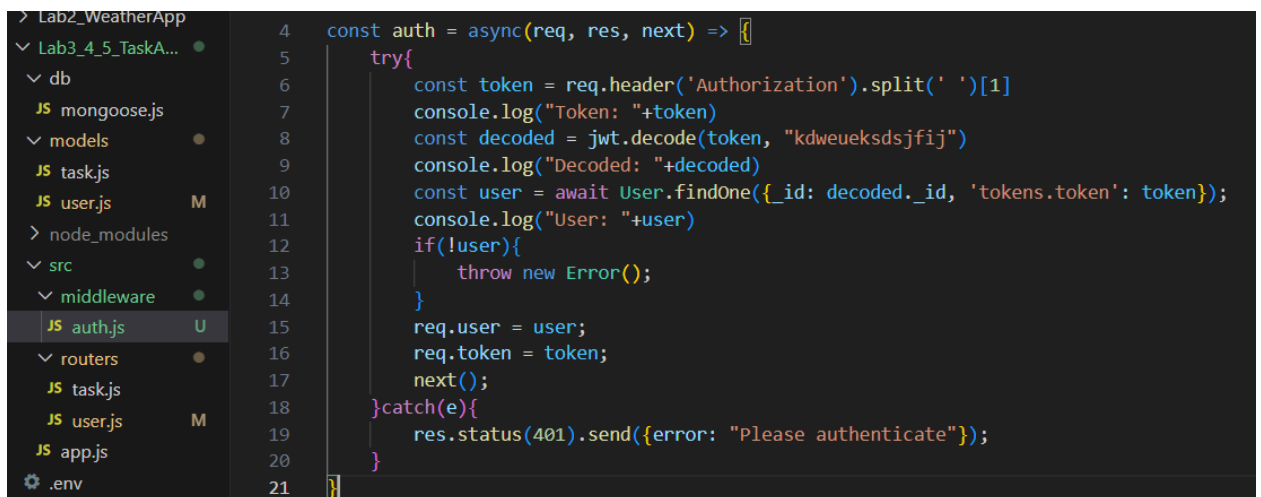


Рис. 23. Результат

Крок 3.2

- Створіть запит для перегляду даних про авторизованого користувача

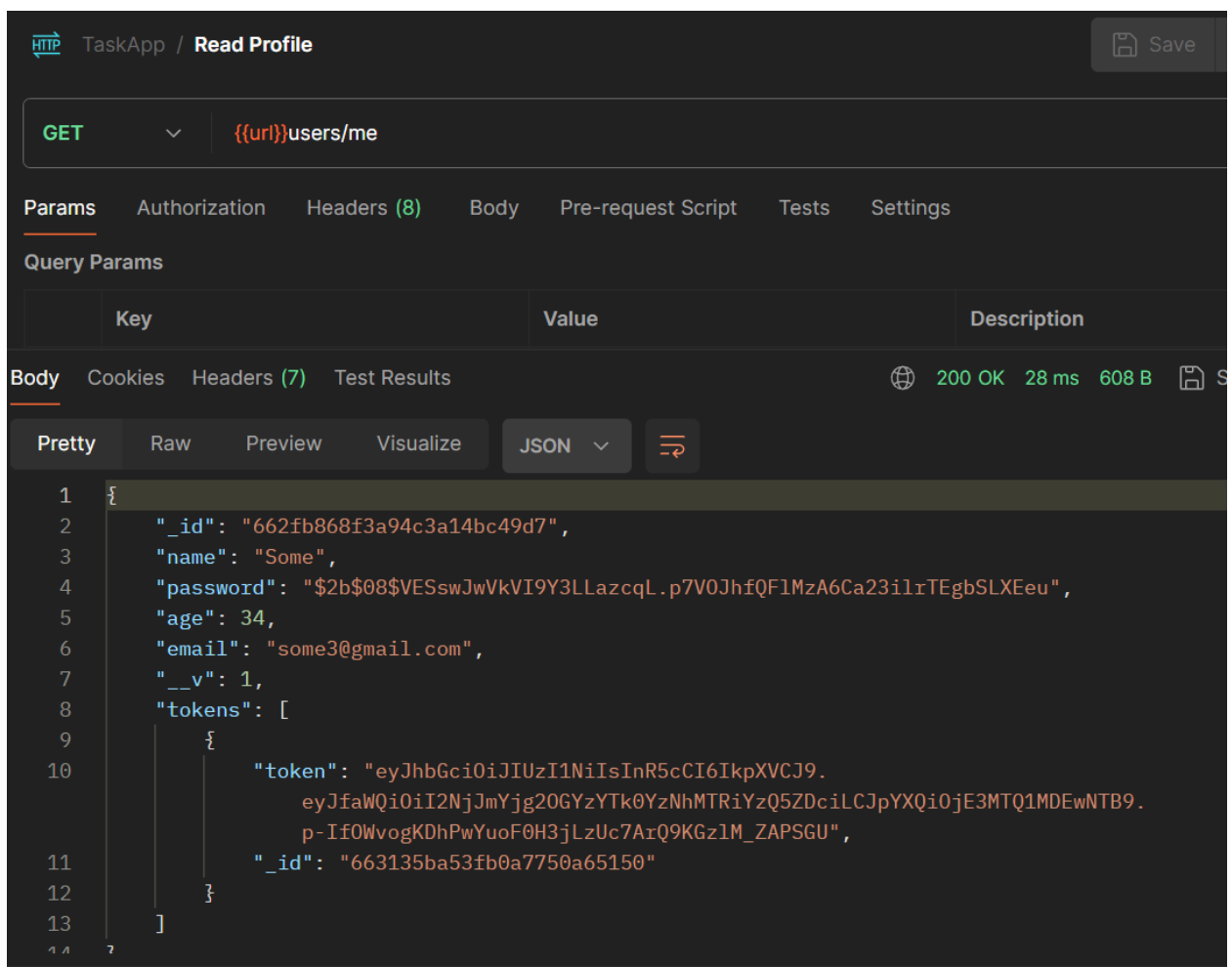
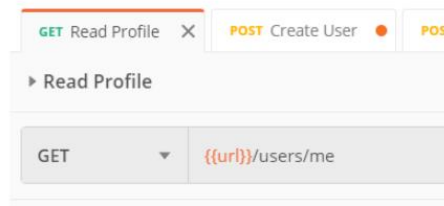


Рис. 24. Результат

Крок 3.3

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

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

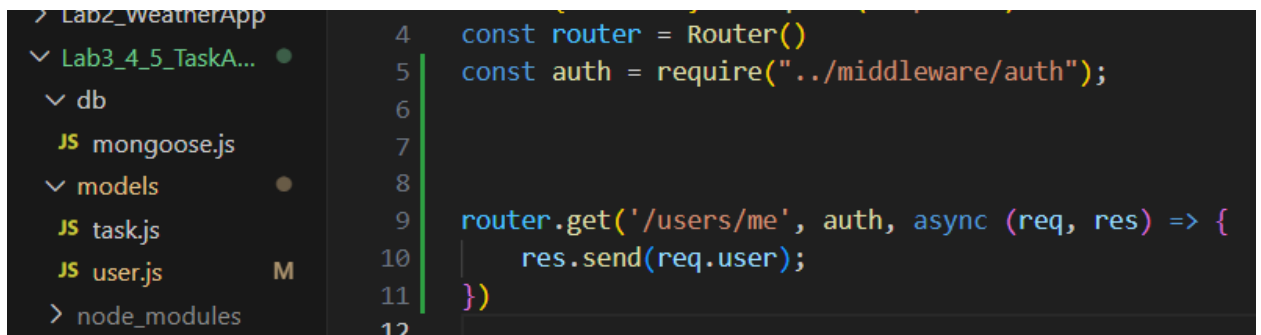


Рис. 25. Результат

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

Крок 4.1

- Створити POST-запит users/logout

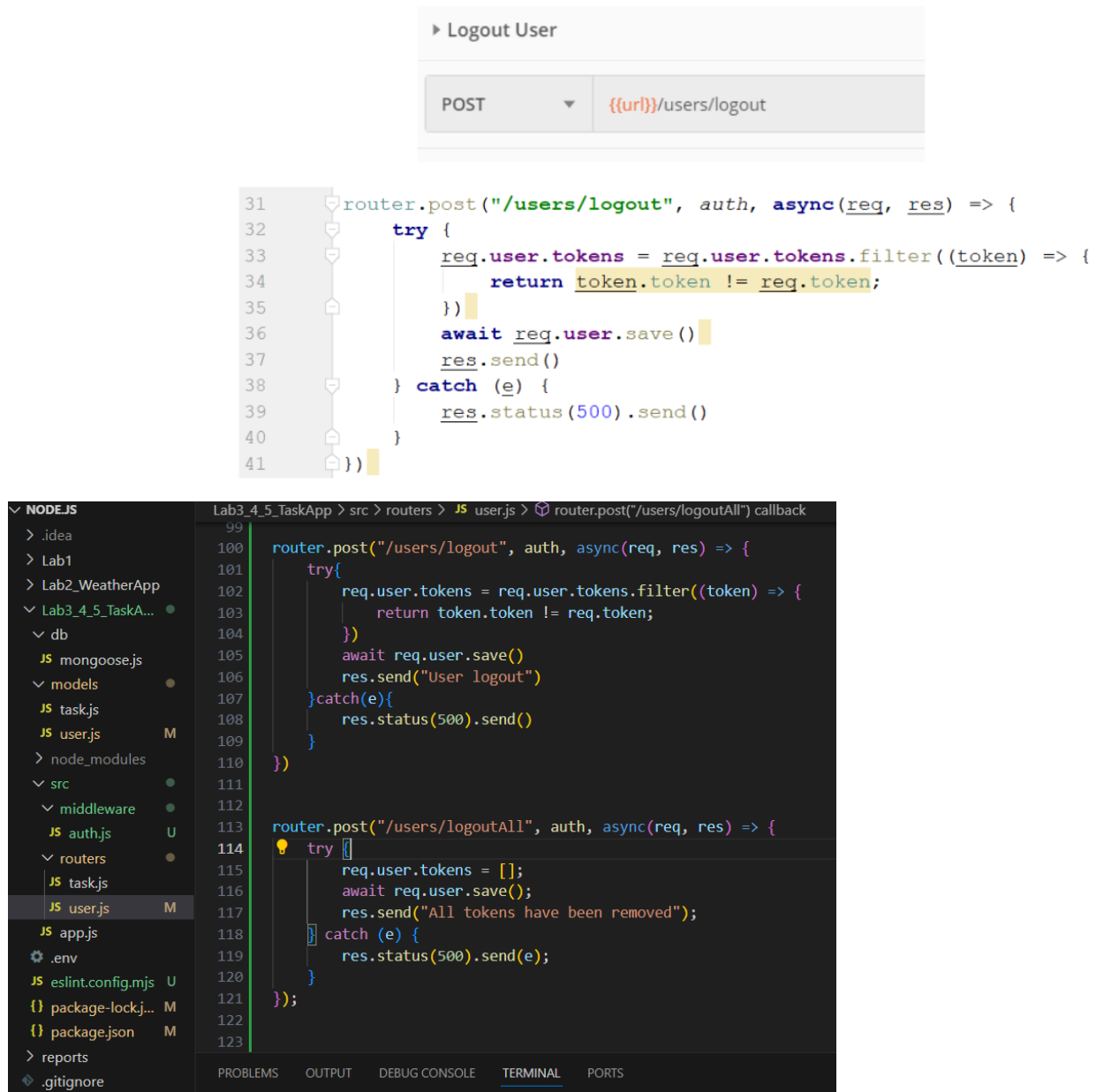


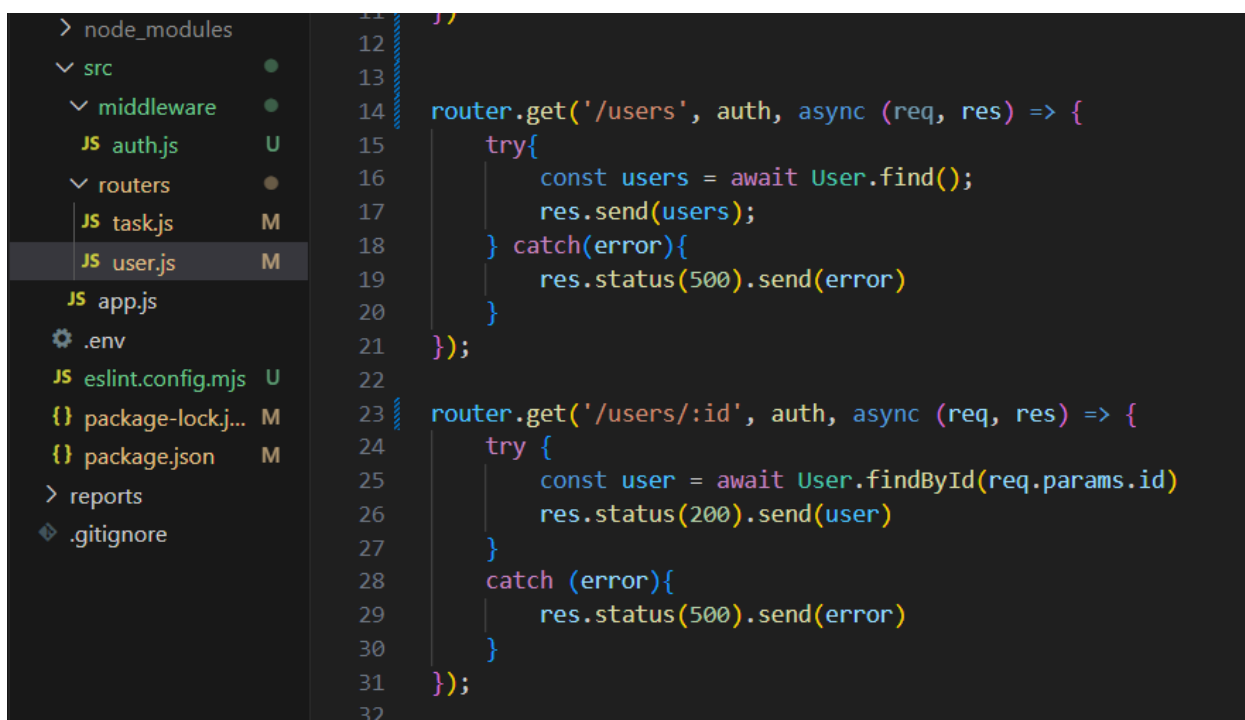
Рис. 26. Результат

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



```
11  })
12
13
14  router.get('/users', auth, async (req, res) => {
15    try{
16      const users = await User.find();
17      res.send(users);
18    } catch(error){
19      res.status(500).send(error)
20    }
21  });
22
23  router.get('/users/:id', auth, async (req, res) => {
24    try {
25      const user = await User.findById(req.params.id)
26      res.status(200).send(user)
27    }
28    catch (error){
29      res.status(500).send(error)
30    }
31  });
32
```

Рис. 27. Результат зміни коду

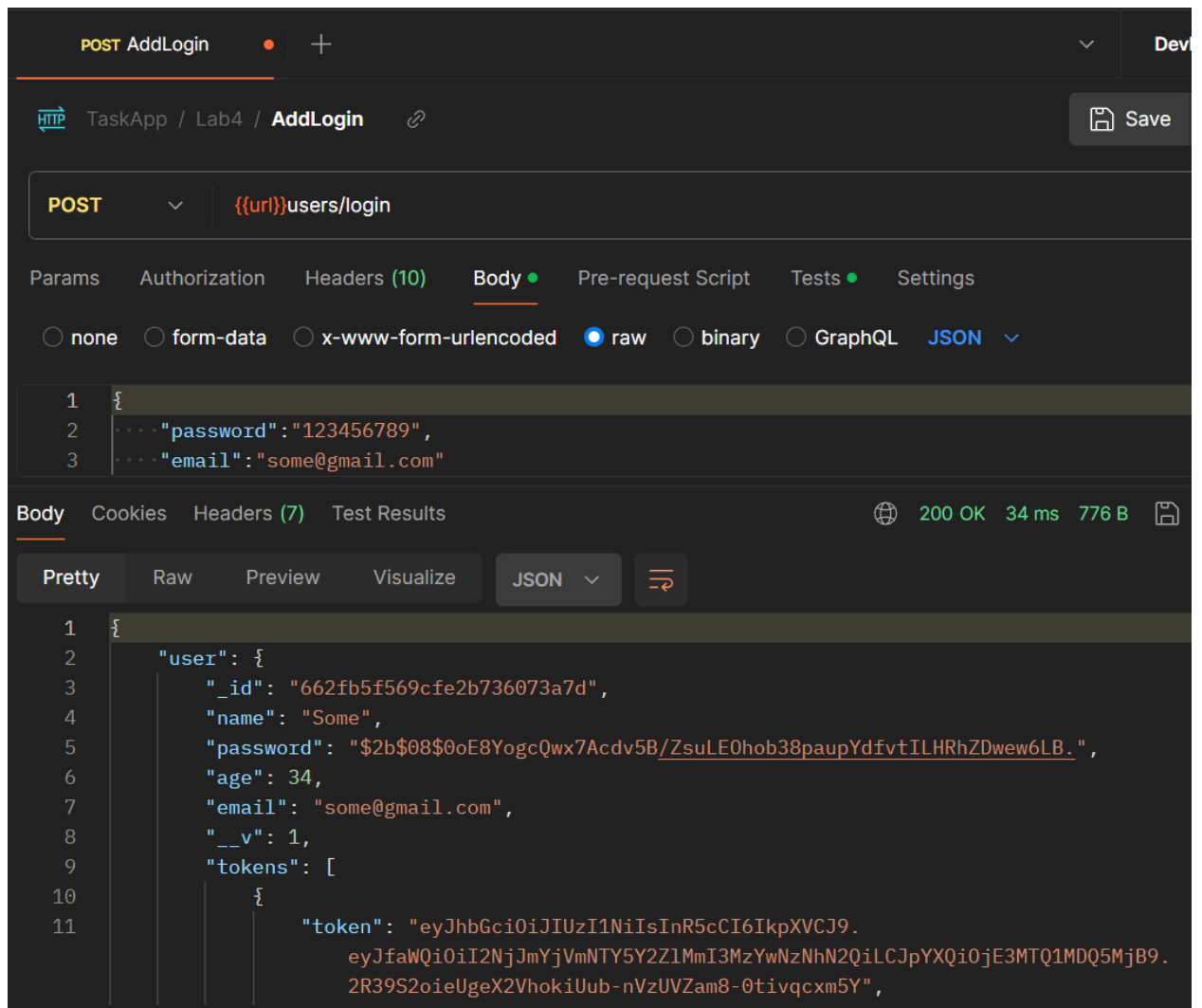


Рис. 28. Вхід

		Клосович І.А.			ДУ «Житомирська політехніка».24.121.13.000 - Лр4	Арк.
		Сидорчук В.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

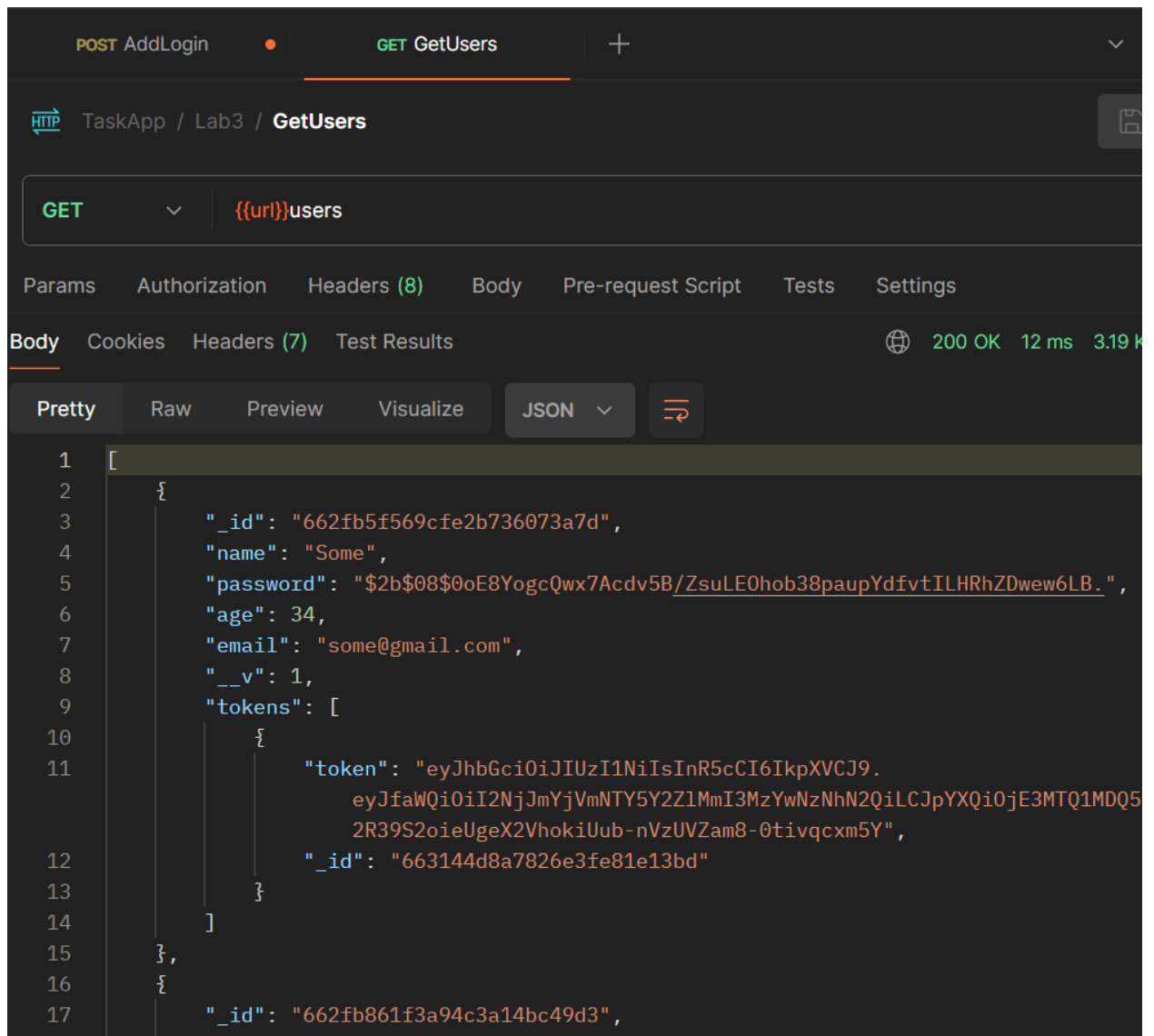


Рис. 29. Запит на перелік всіх users

		Клосович І.А.			ДУ «Житомирська політехніка».24.121.13.000 - Лр4	Арк.
		Сидорчук В.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

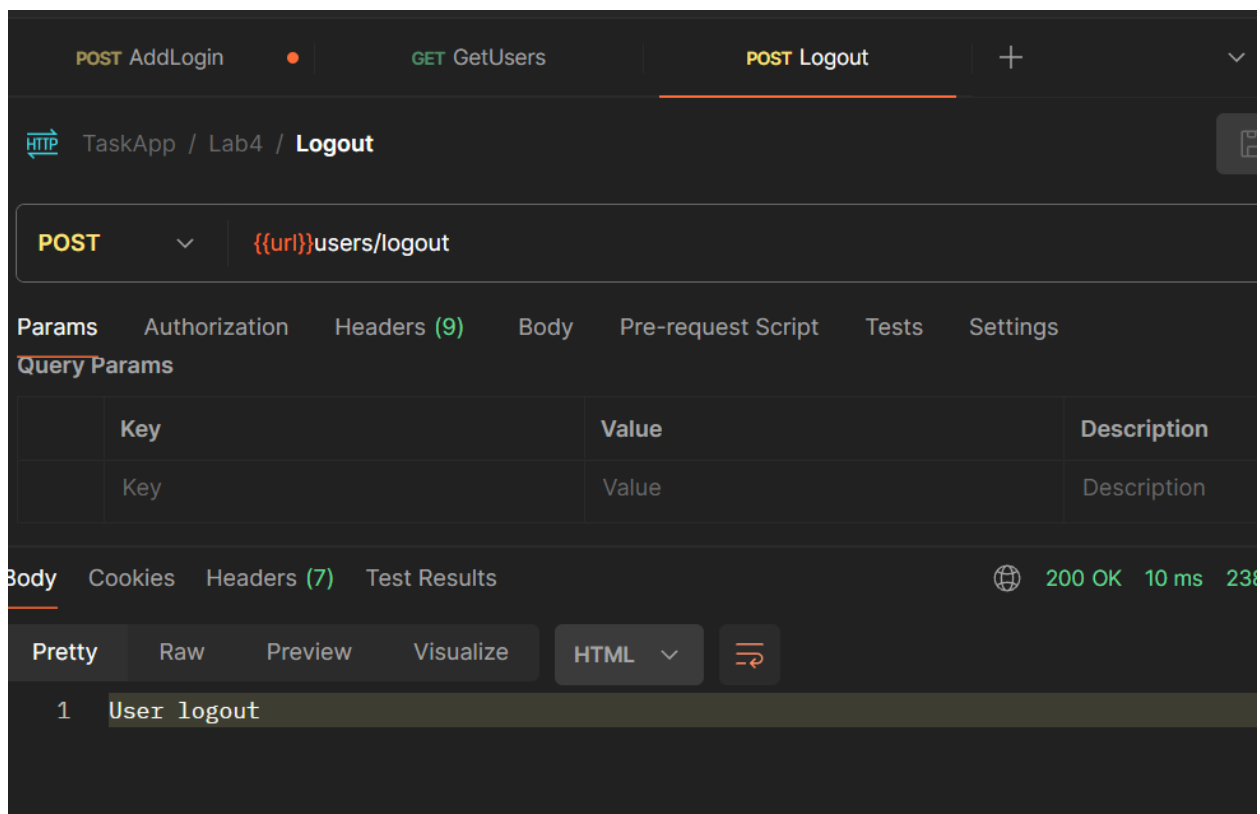


Рис. 30. Вихід користувача

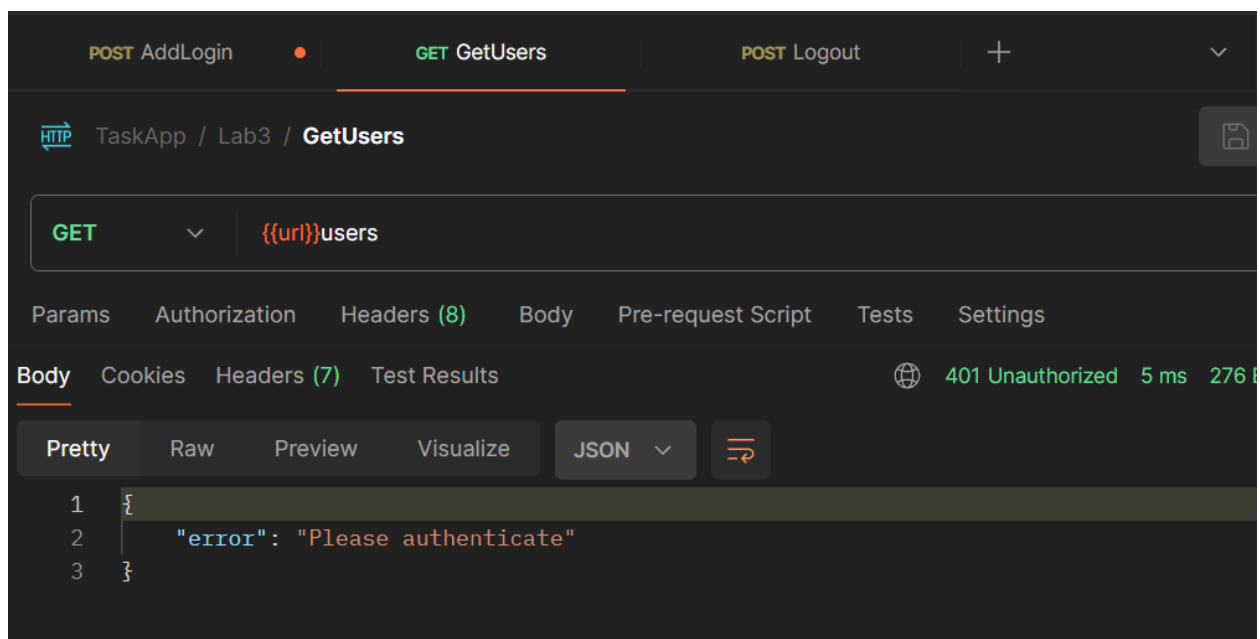


Рис. 31. Помилка виводу всіх користувачів, бо user не автентифікований

Висновок: на лабораторному занятті ми ознайомились з API Authentication and Security.