**ЛАБОРАТОРНА РОБОТА № 3**

**API AUTHENTICATION AND SECURITY**

**ХІД РОБОТИ**

Здійсніть шифрування паролів при редагуванні чи додаванні користувача.

let userSchema = new ***mongoose***.Schema({  
 name: {  
 type: ***String***,  
 required: true,  
 trim: true  
 },  
 age:{  
 type:***Number***,  
 validate(value){  
 if(value < 0){  
 throw new ***Error***("Age must be a positive number")  
 }  
 }  
 },  
 email: {  
 type: ***String***,  
 required: true,  
 trim: true,  
 lowercase: true,  
 unique: true,  
 validate(value) {  
 if(***validator***.isEmail(value) == false){  
 throw new ***Error***("Email is invalid")  
 }  
 }},password: {  
 type: ***String***,  
 required: true,  
 trim: true,

validate(value) {  
 if(value.length < 7){  
 throw new ***Error***("Довжина поля не повинна бути меншою ніж 7")  
 } if(value.indexOf("password") >= 0){  
 throw new ***Error***("Пароль не повинен містити слово \*password\*")  
 }  
 }  
 },  
 tokens: [{  
 token: {  
 type: ***String***,  
 required: true,  
 }  
 }]  
}  
)

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

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

const bcrypt = require('bcrypt');

userSchema.pre('save', async function(next) {  
 const user = this;  
 if(user.isModified('password')){  
 user.password = await bcrypt.hash(user.password, 8);  
 }  
 next();  
});

Створіть обробку на редагування даних (PUT)

3) В обробнику для редагування даних в маршрутизаторі user замість методу findByIdAndUpdate() рекомендується реалізувати save() наступним способом:

Лістинг:

***router***.use("/user-update/:id",async (req, res) =>{  
 const user = await ***userModel***.findById(req.params.id);  
 try {  
 const updates = ['name', 'email', 'password','age'];  
 updates.forEach((update) => user[update] = req.body[update]);  
 await user.save();  
 res.send(user);  
 }catch (e) {  
 res.status(500).send(e);  
 }  
});

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

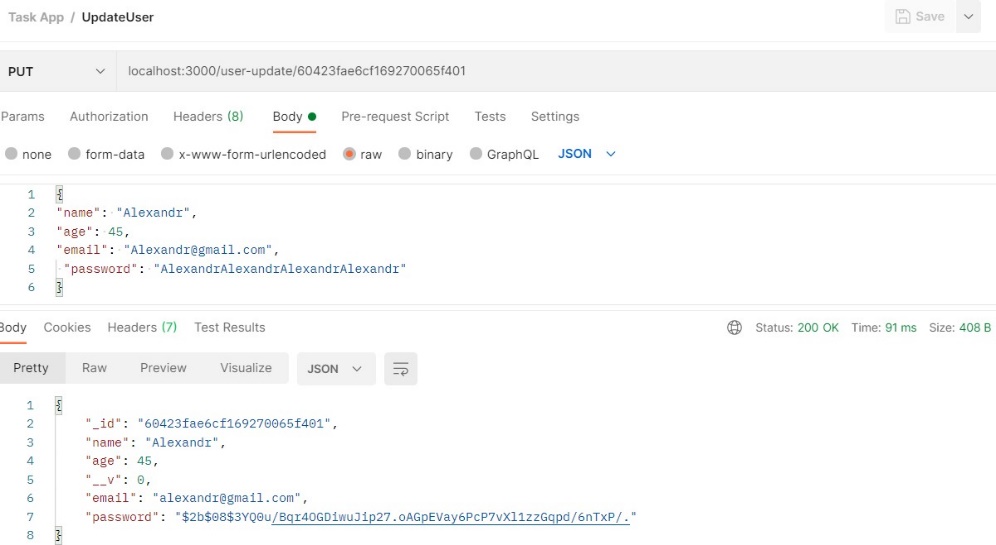


Рис 1. Демонстрація шифрування пароля.

**Завдання.**

Здійснимо запит на авторизацію користувача.

Крок 1.

1. Створюємо та зберігаємо запит .

**Результат виконання:**

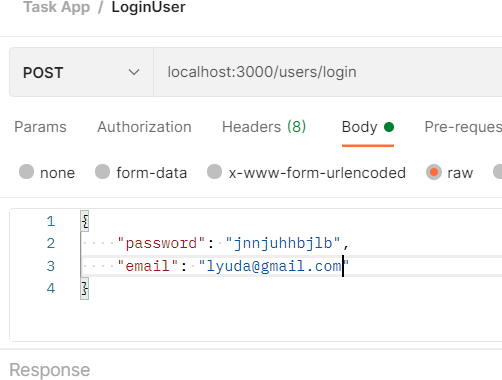


Рис 2. Створення запита.

Крок 2.

Для моделі даних реалізуємо статичний метод findOneByCredentials(), який перевірятиме правильність авторизації.

userSchema.statics.findOneByCredentials = async (email, password) => {  
 const user = await module.exports.findOne({email});  
  
 if (!user){  
 throw new ***Error***('Incorrect email!');  
 }  
  
 const isMatch = await bcrypt.compare(password, user.password);  
 if (!isMatch){  
 throw new ***Error***('Incorrect password');  
 }  
 return user;  
};

Крок 3.

Реалізуйте обробку запиту за маршрутом /users/login з використанням методу findOneByCredentials().

Лістинг:

***router***.post("/users/login", async (req, res) => {  
 try{  
 const user = await ***userModel***.findOneByCredentials(req.body.email, req.body.password);  
 const token = await user.generateAuthToken();  
 res.send({user, token});  
 }  
 catch (e) {  
 res.status(400).send(e.message);  
 }  
});

Крок 4.

Перевірте виконання запиту при вірно і невірно введених даних.

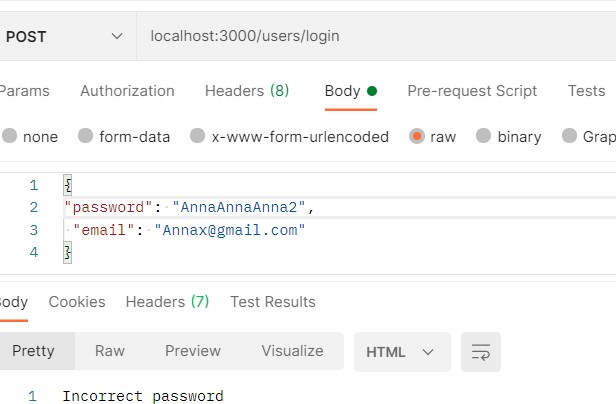
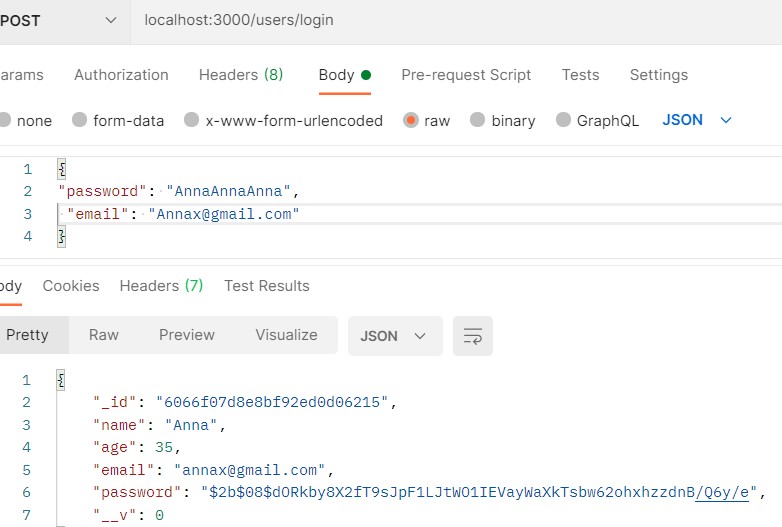


Рис 3. Виконання запиту.

**Завдання.**

Реалізувати генерацію токену та його збереження в БД при реєстрації та авторизації користувачів. Крок 1 В схемі даних моделі створимо нову властивість tokens – масив токенів.

Лістинг:

tokens: [{  
 token: {  
 type: ***String***,  
 required: true,  
 }  
}]

Крок 2.

Для схеми даних моделі створимо метод для генерування токену:

Лістинг:

userSchema.methods.generateAuthToken = async function () {  
 const user = this;  
 const token = ***jwt***.sign({\_id: user.\_id.toString()}, 'kdweueksdsjfij');  
 user.tokens = user.tokens.concat({token});  
 await user.save();  
 return token;  
};

Крок 3.

Створюємо токен при успішній авторизації:

const token = await user.generateAuthToken();  
res.send({user, token});

Крок 4.

Переконайтеся у створенні токенів для запитів реєстрації та логіну.

в БД:

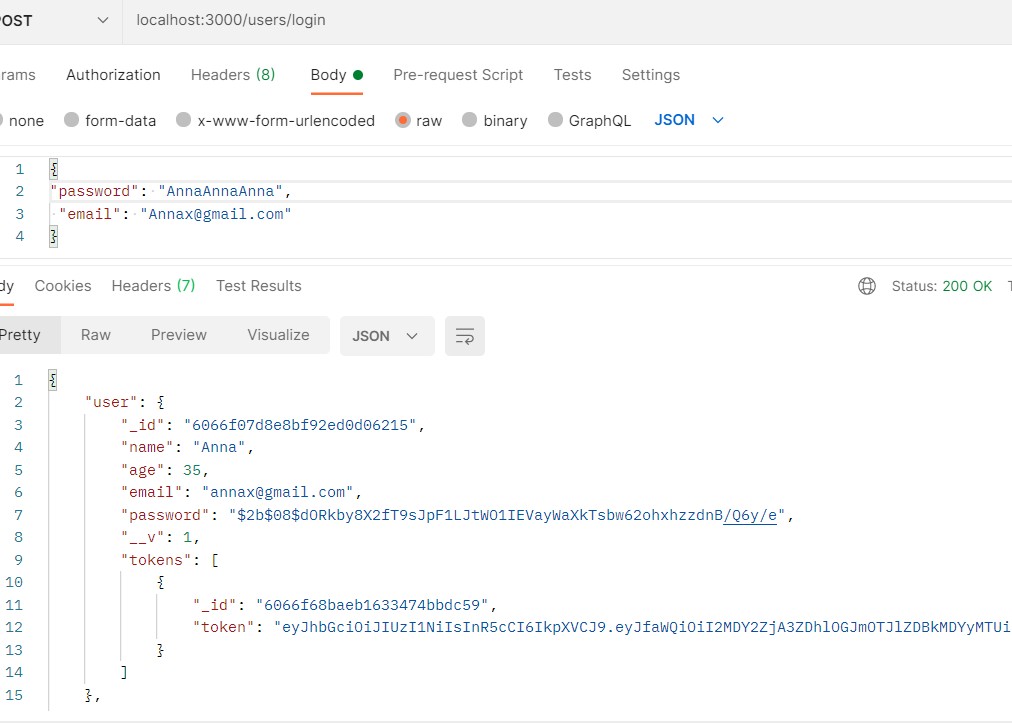
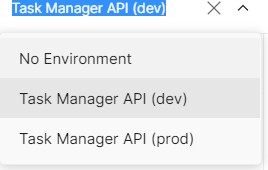


Рис 4. Результат виконання в БД.

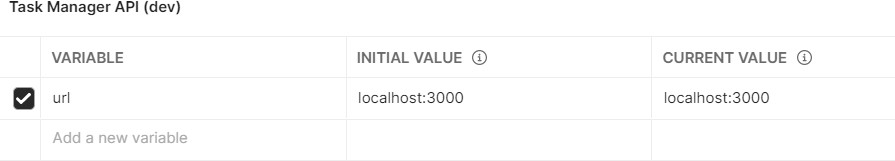
**Завдання.**

Використання згенерованого токену для перегляду даних про користувача Крок1 В Postman створіть два середовища Task Manager API (dev) і Task.



Крок 2

В середовищі Task Manager API (dev) створіть змінну url із значенням localhost:3000 Використайте змінну url у всіх існуючих запитах.



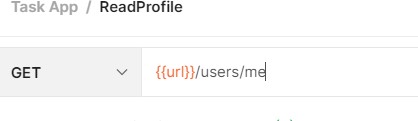
Крок 3.

Лістинг:

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, 'kdweueksdsjfij');  
 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 5.

Створіть запит для перегляду даних про авторизованого користувача.



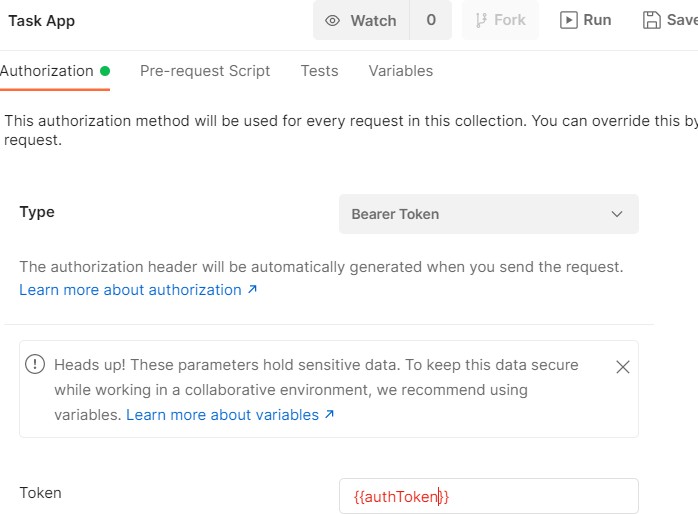
Крок 6.

Здійсніть обробку даного запиту з попереднім виконанням middleware-функції auth.

***router***.get('/users/me', auth, async (req, res) => {  
  
 const users = req.user;  
 await users.populate('tasks').execPopulate();  
 let data = users.name + '\n'+ users.age+'\n'+ users.email+'\n Мої tasks:\n'+ users.tasks;  
 res.send(data);  
});

Крок 7 .

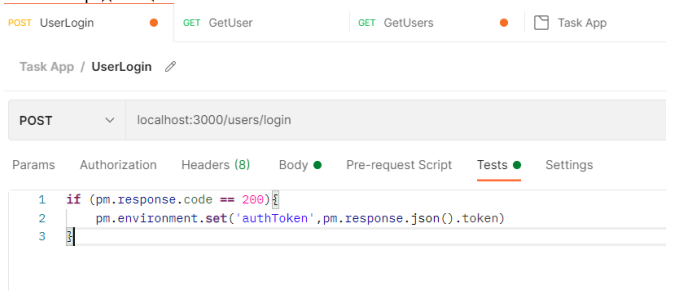
1) На рівні колекції (батьківський об’єкт) для наших запитів створити метод авторизації Bearer Token і в полі Token задати змінну оточення {{authToken}}.



Крок 9.

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

середовища.



Крок 10.

Створити POST-запит users/logout.

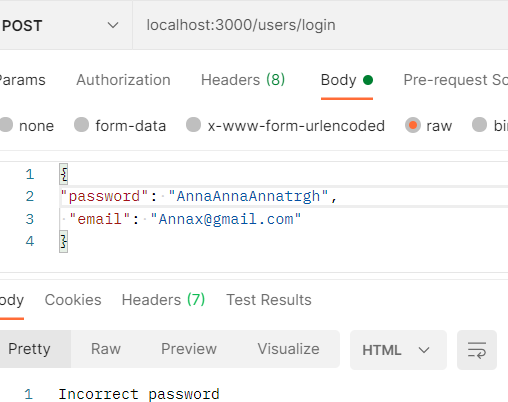
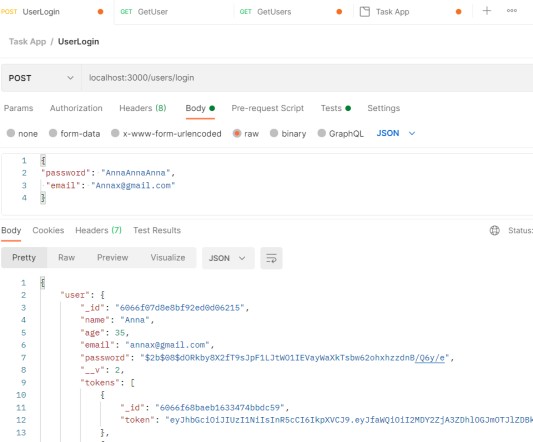
Лістинг:

***router***.post('/users/logout', auth, async (req, res) => {  
 try {  
 req.user.tokens = req.user.tokens.filter((token) => {  
 return token.token != req.token;  
 });  
 await req.user.save();  
 res.send();  
 } catch (e) {  
 res.status(500).send(e.message);  
 }  
});

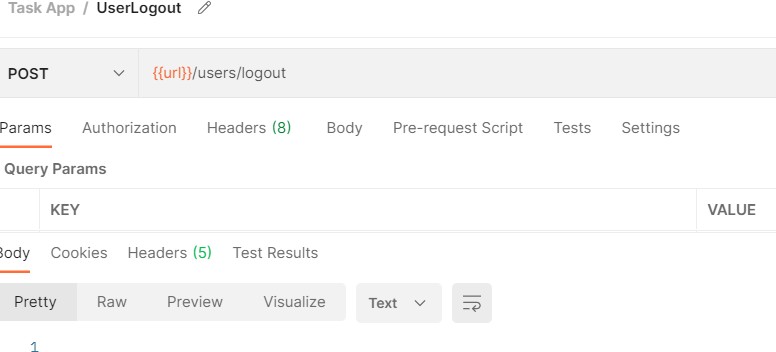
**Протестуйте виконання.**

Завдання 4.

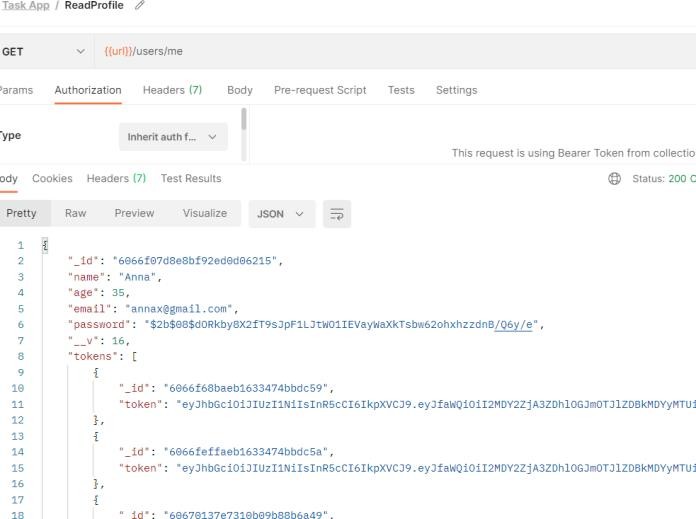
Успішна/не успішна авторизація.



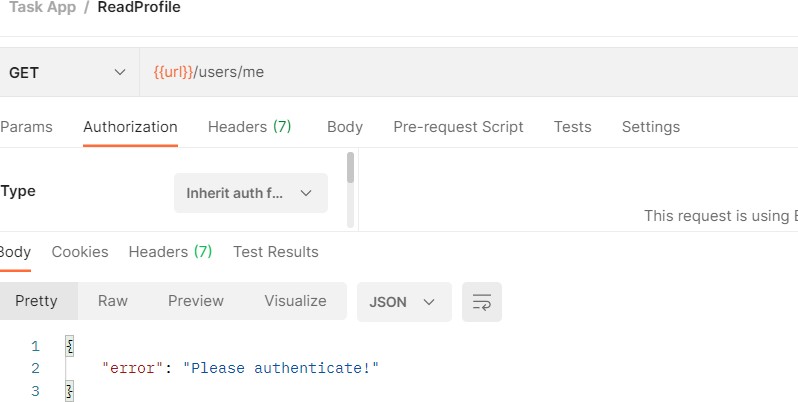
Успішний вхід.



Перехід на сторінку авторизованого користувача.



Помилка при переході на сторінку неавторизованого користувача.



**Висновок:** на даній лабораторній роботі ознайомився з API Authentication and

Security.