

## Лабораторна робота №4

### API Authentication and Security

#### Хід роботи :

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

#### Схема даних і модель

```
const User : Schema<any, Model<...>, {...}, {...}, {...}, {...}, DefaultSchemaOptions, = new mongoose.Schema({
  name: {type: String...},
  age: {type: Number...},
  email: {type: String...},
  password: {type: String...}
})
const someUser : Model = mongoose.model("User", User)
module.exports = someUser
```

#### Шифрування паролю

```
8
new *
9 User.pre( method: 'save', fn: async function(next : CallbackWithoutResultA
0   const user = this;
1   if(user.isModified( path: 'password')){
2     user.password = await bcrypt.hash(user.password, salt: 8);
3   }
4   next();
5 }
```

Створіть обробку на редагування даних (PATCH)

					ЖИТОМИРСЬКА ПОЛІТЕХНІКА.21.121.01.000 – Лр.4		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Алієв О.Є			Звіт з лабораторної роботи №4	Літ.	Арк.
Перевір.		Сидорчук В.О					1
Реценз.						ФІКТ, гр.ІПЗ-22-1	
Н. Контр.							
Зав.каф.							

```

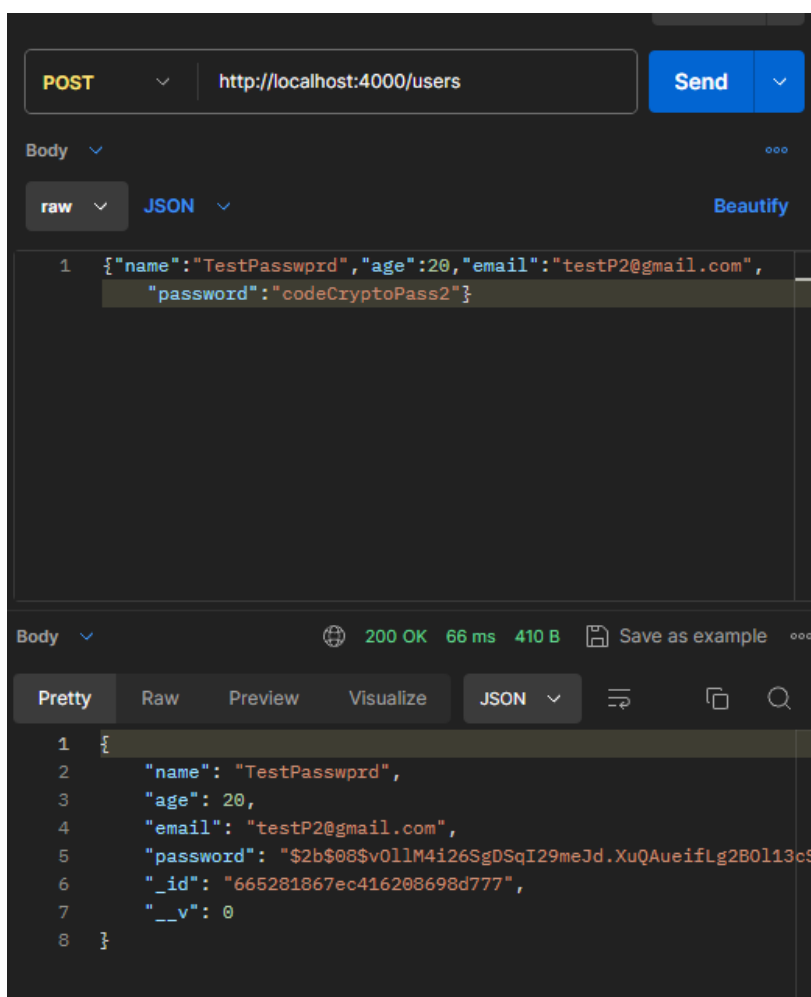
router.patch( path: '/users/:id', handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj>
  try {
    const { id } = req.params
    const user : Query = await User.findById(id)
    if (!user) {
      throw new Error("Not existing")
    }

    const fields : string[] = ['name', 'password', 'age', 'email']
    fields.forEach(field : string => {
      if (req.body[field]) {
        user[field] = req.body[field]
      }
    })

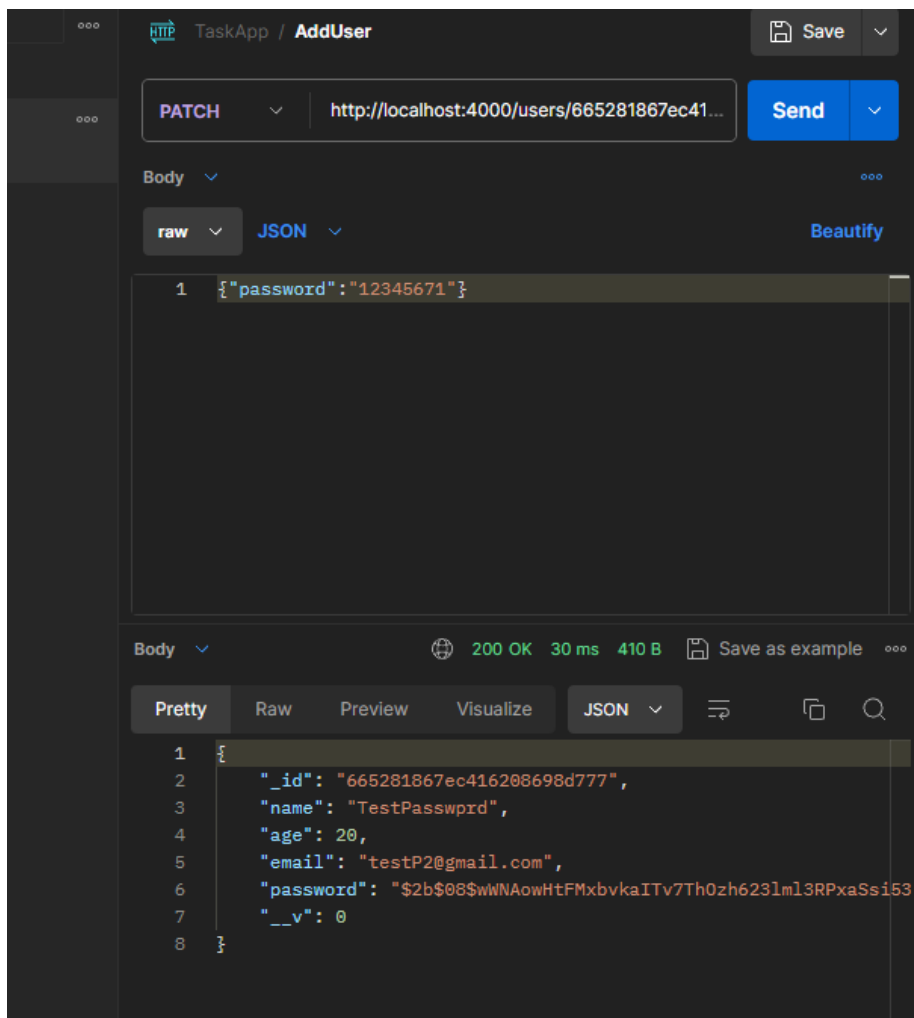
    await user.save()
    res.send(user)
  } catch(e) {
    next(e)
  }
})

```

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

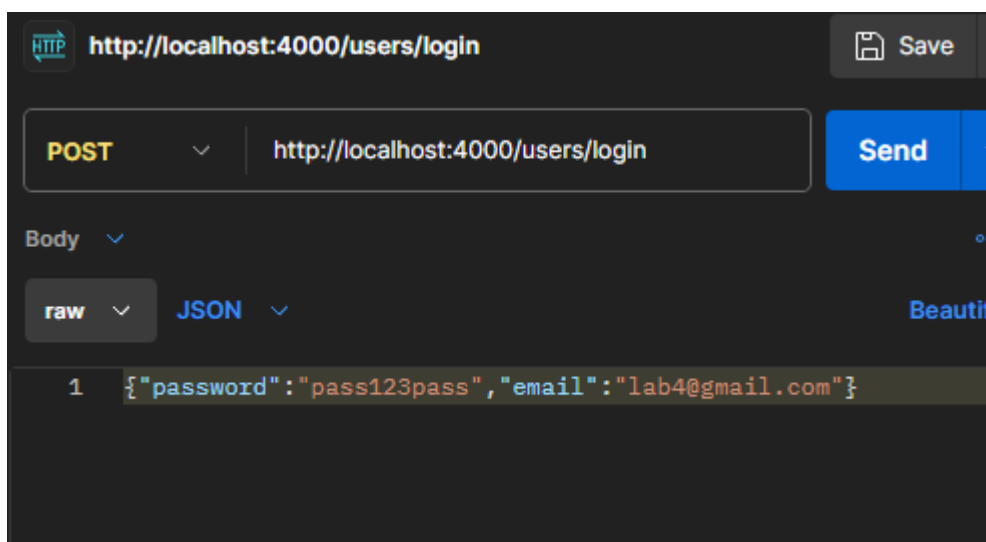


					ЖИТОМИРСЬКА ПОЛІТЕХНІКА.20.121.01.000 – Лр.4	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		2



2. Створити запит /users/login на вхід.

Створіть запит /users/login



					ЖИТОМИРСЬКА ПОЛІТЕХНІКА.20.121.01.000 – Лр.4	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		3

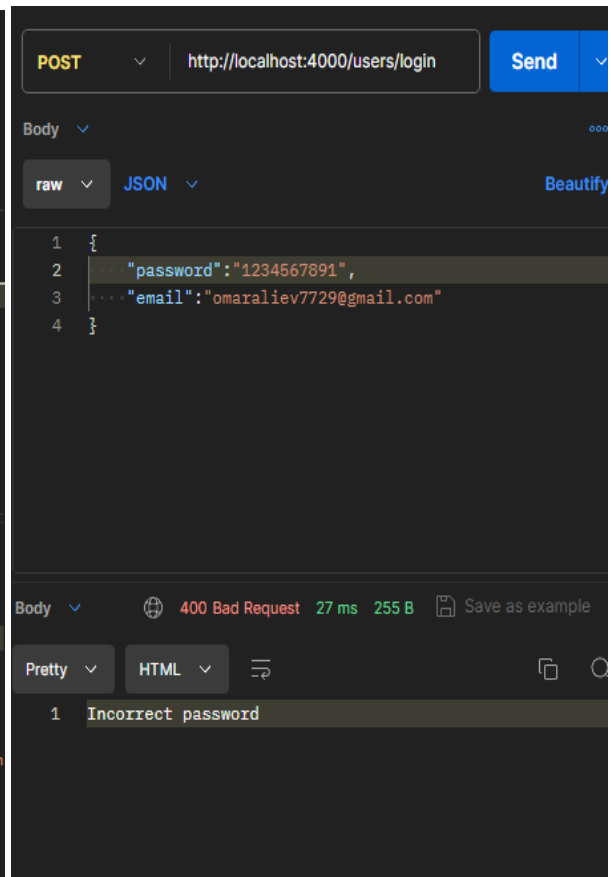
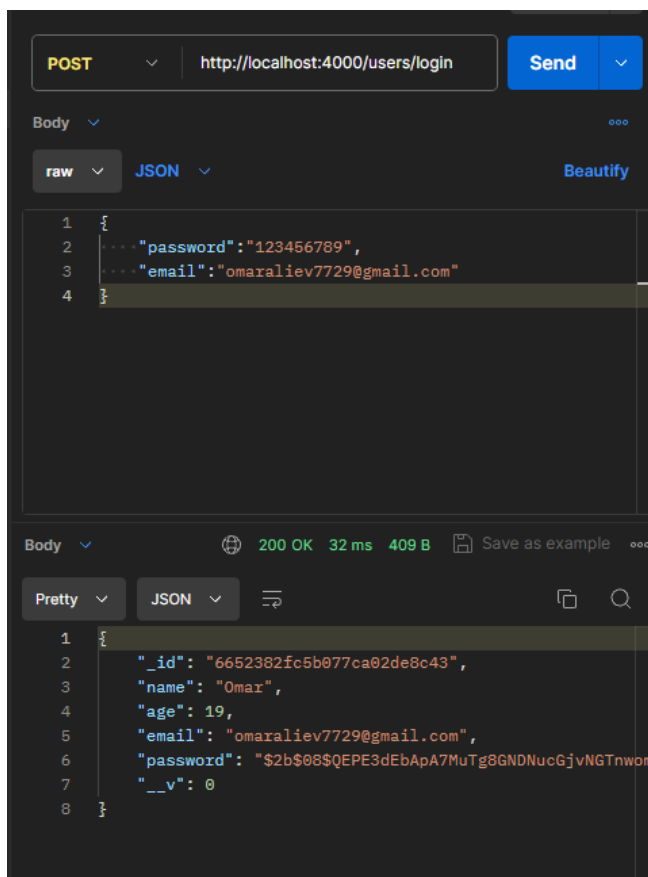
## Створіть метод для аутентифікації

```
User.statics.findOneByCredentials = async (email, password) : Promise<any> => {
  const user = await User.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
}
```

## Викличте метод аутентифікації

```
new *
router.post( path: '/users/login', handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj>
  try {
    const user = await User.findOneByCredentials(req.body.email, req.body.password)
    res.send(user);
  } catch(e) {
    res.status( code: 400).send();
  }
})
```

## Протестуйте аутентифікацію



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

```
lowerCase: true,  
unique: true,  
validate: {  
  validator: (value) => {  
    return validator.isEmail(value);  
  },  
  message: 'Error: Email is invalid'  
},  
},  
tokens: [{  
  token: {  
    type: String,  
    required: true  
  }  
}]  
}]
```

Створимо метод для генерації токена

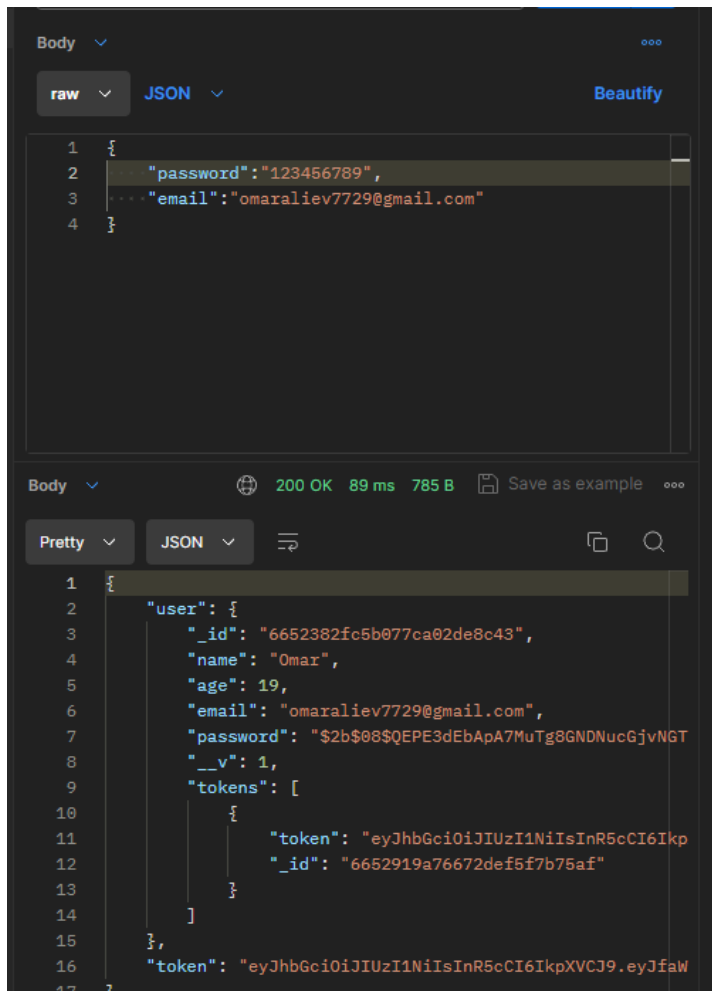
```
User.methods.generateAuthToken = async function () : Promise<any> {  
  const user = this  
  const token = jwt.sign( {payload: { _id: user._id.toString() }, secretOrPrivateKey: "kdweueksdsjfi"} )  
  user.tokens = user.tokens.concat({ token })  
  await user.save()  
  return token  
}
```

Забезпечимо генерацію токена при успішній аутентифікації

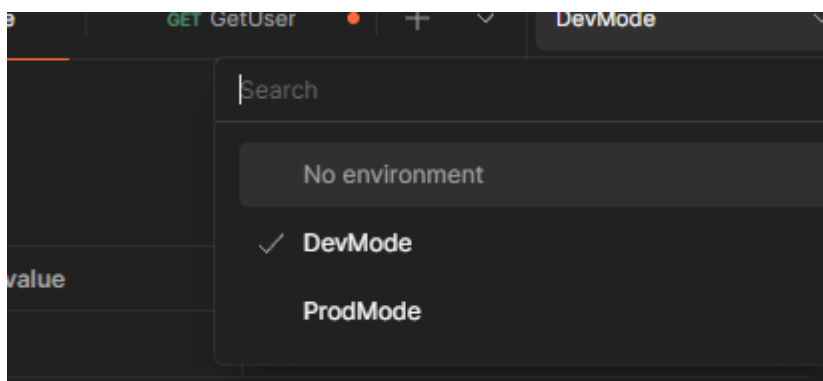
```
const user = await User.findOne({username: req.body.username});  
const token = await user.generateAuthToken();  
res.send( {body: {user, token}});  
} catch(e) {
```

Тестуємо

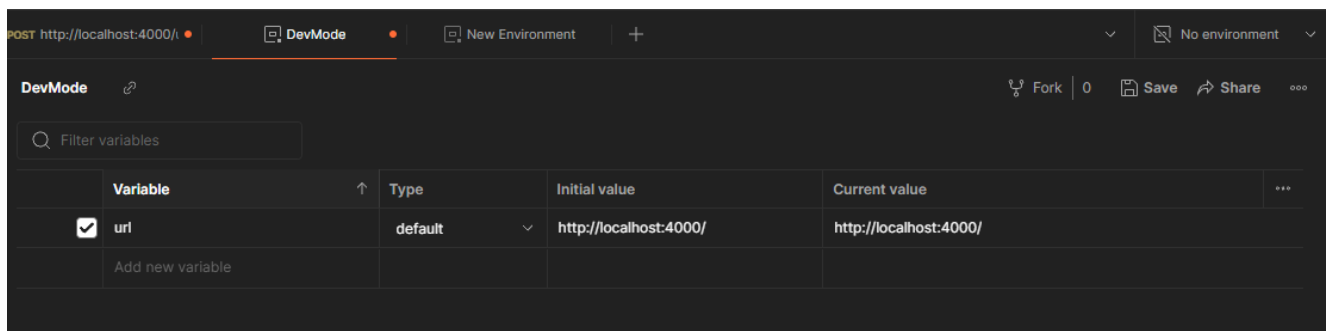
					ЖИТОМИРСЬКА ПОЛІТЕХНІКА.20.121.01.000 – Лр.4	Арк.
						5
Змн.	Арк.	№ докум.	Підпис	Дата		



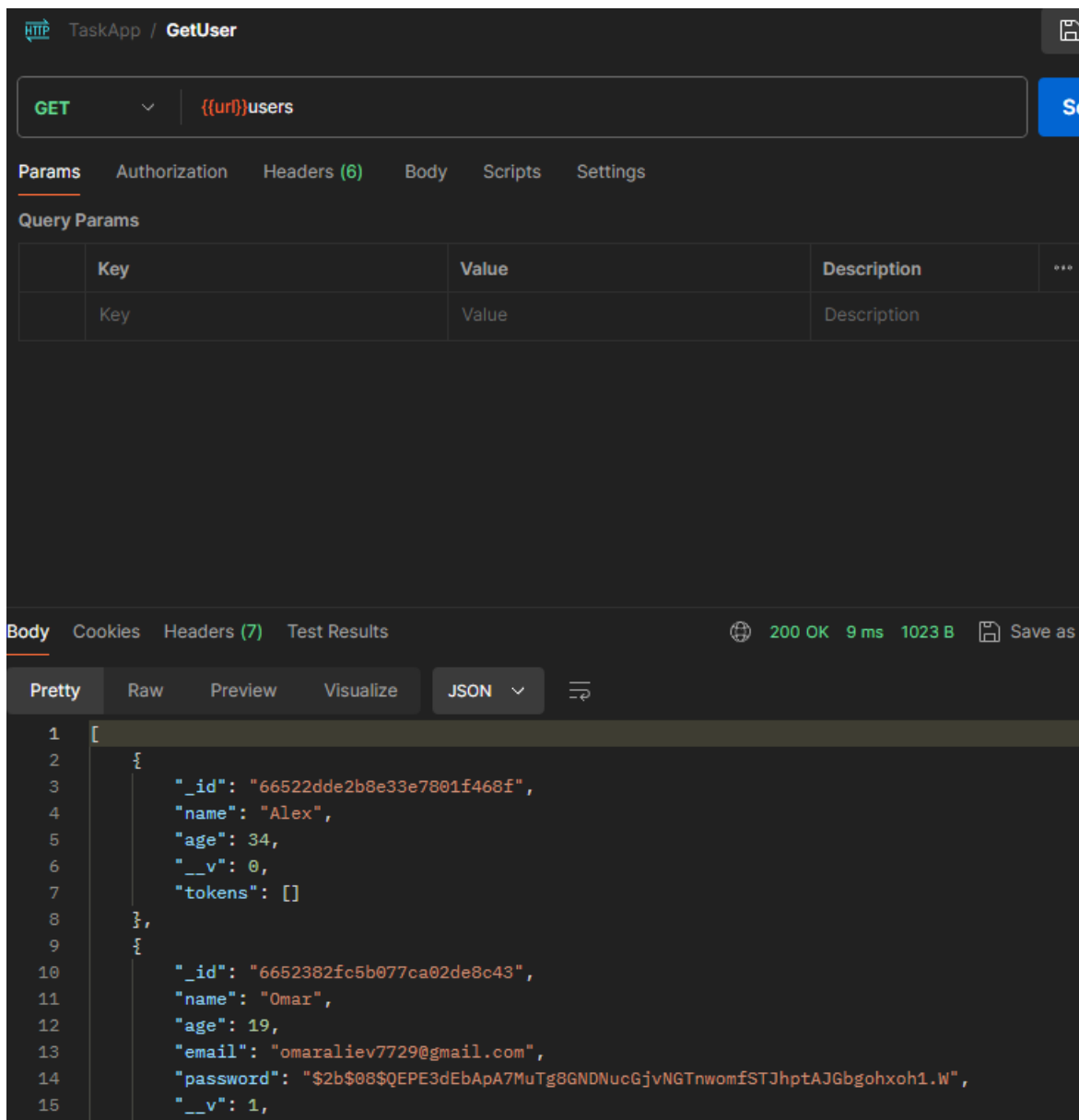
Середовище розробки і робоче середовище



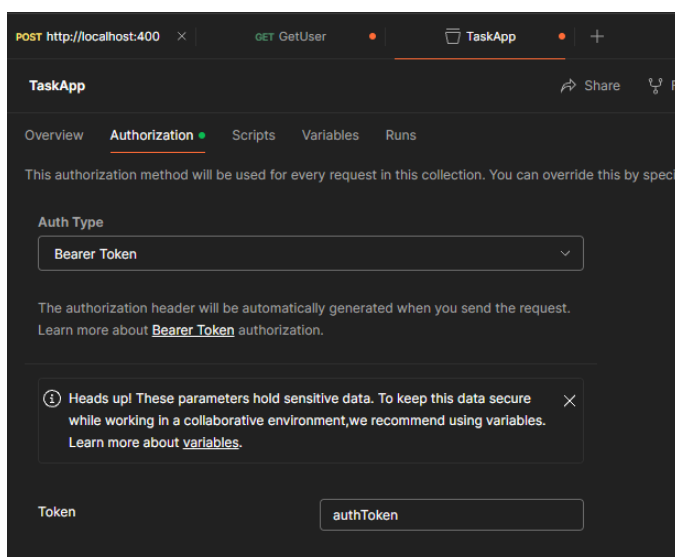
Змінна оточення *url*



					ЖИТОМИРСЬКА ПОЛІТЕХНІКА.20.121.01.000 – Лр.4	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		6

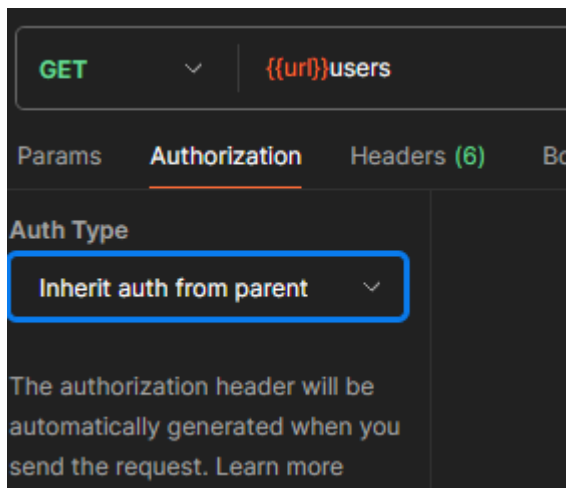


В Postman на рівні папки для наших запитів створити метод авторизації Bearer Token і в полі Token задати змінну оточення {{authToken}}

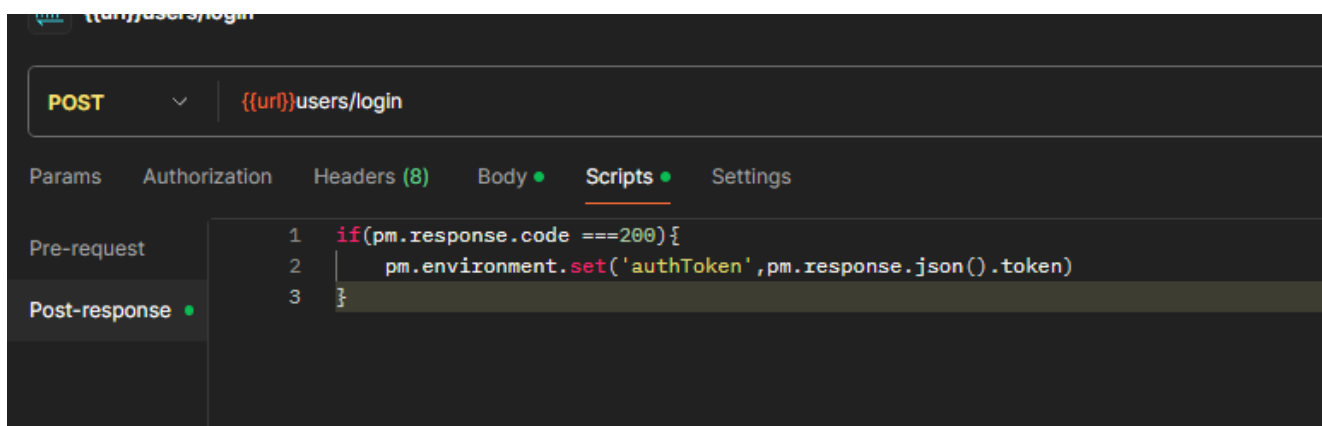


					ЖИТОМИРСЬКА ПОЛІТЕХНІКА.20.121.01.000 – Лр.4	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

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



Автоматизуємо передачу значення змінній authToken



### 3. Примініти авторизацію

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

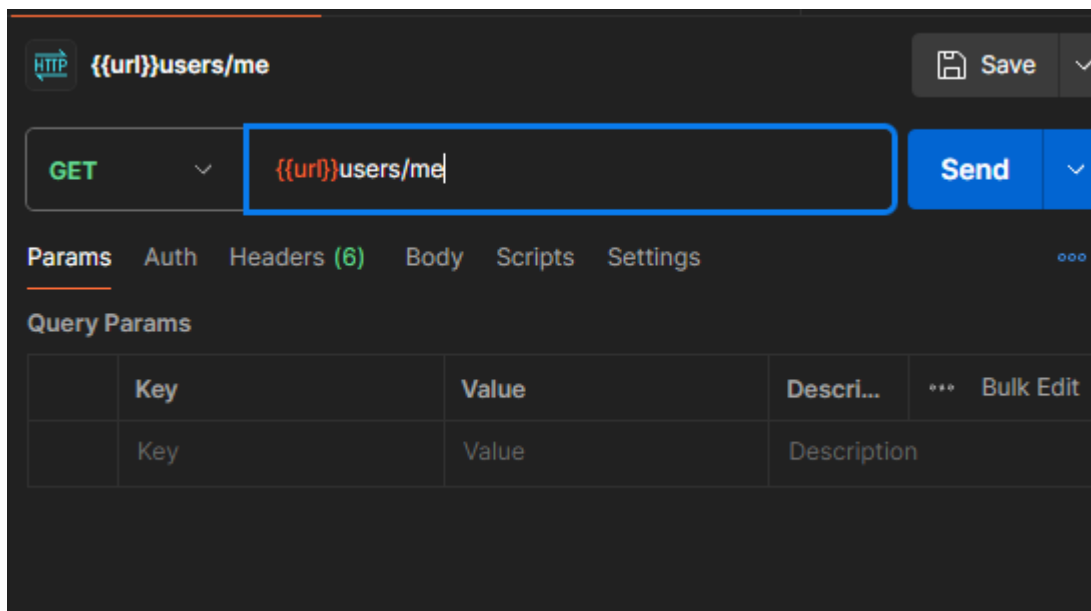


```

1 usage new *
2
3
4 const auth = async (req, res, next) : Promise<void> => {
5     try{
6         const token = req.header('Authorization').split(' ')[1]
7         console.log("Token: "+token)
8         const decoded : { ... } | string = jwt.decode(token, {options: "kdweueksdsjfij"})
9         console.log("Decoded: "+decoded)
10        const user : Query = await User.findOne({_id: decoded._id, 'tokens.token': token})
11        console.log("User: "+user)
12        if(!user){
13            throw new Error();
14        }
15        req.user = user;
16        req.token = token;
17        next();
18    }catch(e){
19        res.status(401).send({error: "Please authenticate"});
20    }
21 }
22
23 module.exports = auth;

```

Створіть запит для перегляду даних про авторизованого користувача



Здійсніть обробку даного запиту з попереднім виконанням middleware-функції auth

```

1 new *
2
3
4 router.get( path: '/users/me', auth, async (req : Request<P, ResBody, ReqBody, ReqQ
5     console.log(req.user)
6     res.send(req.user);
7 })

```

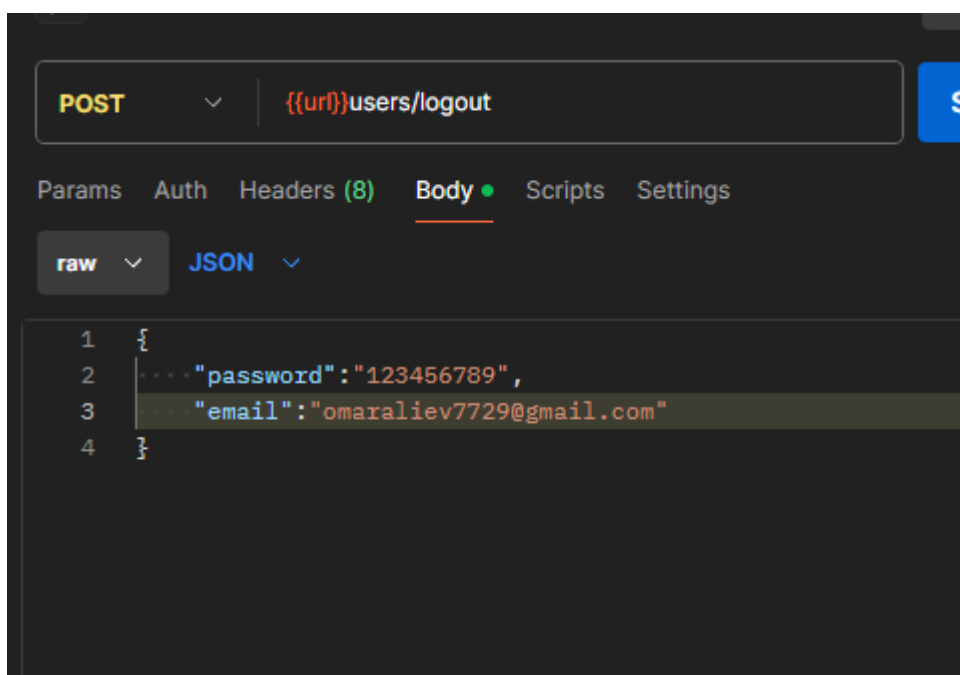
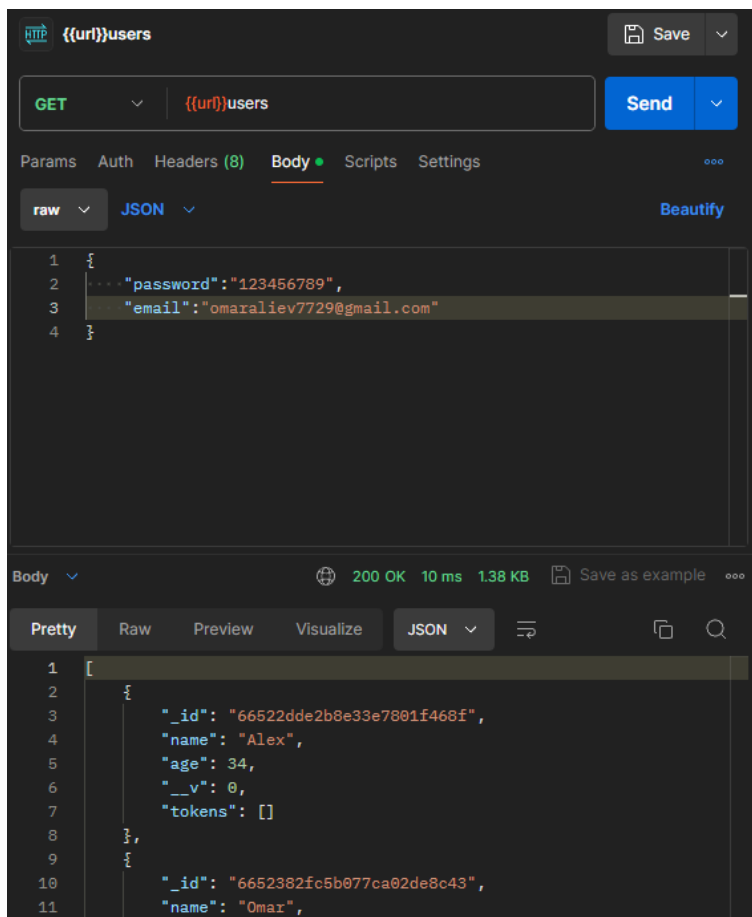
4. Створити запити /users/logout, /users/logoutAll. Вимагає авторизації. Видаляє запис про токен із БД.

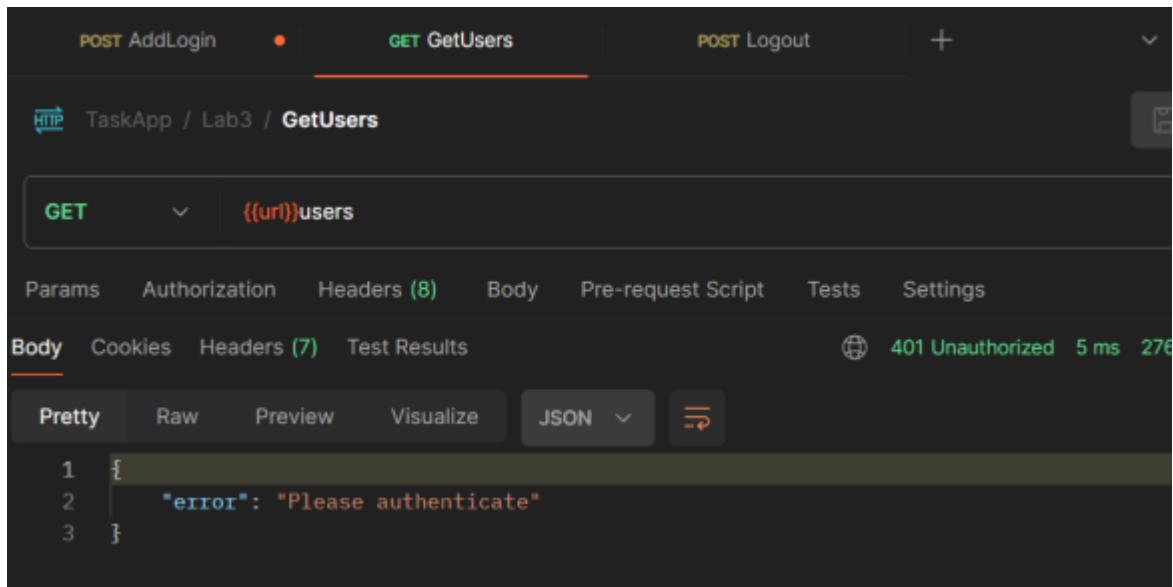
Створити POST-запит users/logout

```
new *
router.post( path: "/users/logout", auth, async(req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , r
    try{
        req.user.tokens = req.user.tokens.filter((token) : boolean => {
            return token.token !== req.token;
        })
        await req.user.save()
        res.send( body: "User logout")
    }catch(e){
        res.status( code: 500).send()
    }
})
```

Протестуйте виконання роботи

The screenshot shows a REST client interface. At the top, the method is set to POST and the URL is {{url}}users/login. The 'Body' tab is selected, showing a raw JSON request body: {"password": "123456789", "email": "omaraliev7729@gmail.com"}. The response is shown in the 'Body' section below, with a status of 200 OK, 76 ms, and 1.15 KB. The response body is a JSON object: {"\_id": "6652382fc5b077ca02de8c43", "name": "Omar", "age": 19, "email": "omaraliev7729@gmail.com", "password": "\$2b\$08\$QEPE3dEbApA7MuTg8GNDNucGjvNGTnwomfSTJhptA:", "\_\_v": 3, "tokens": [{"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfcm9udCI6IjE5OTk0MTUzIiwiaWF0IjoiMTY1MjM4Mjc5In0", "\_id": "6652919a76672def5f7b75af"}]}. The response is formatted as JSON.





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

					ЖИТОМИРСЬКА ПОЛІТЕХНІКА.20.121.01.000 – Лр.4	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		