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)

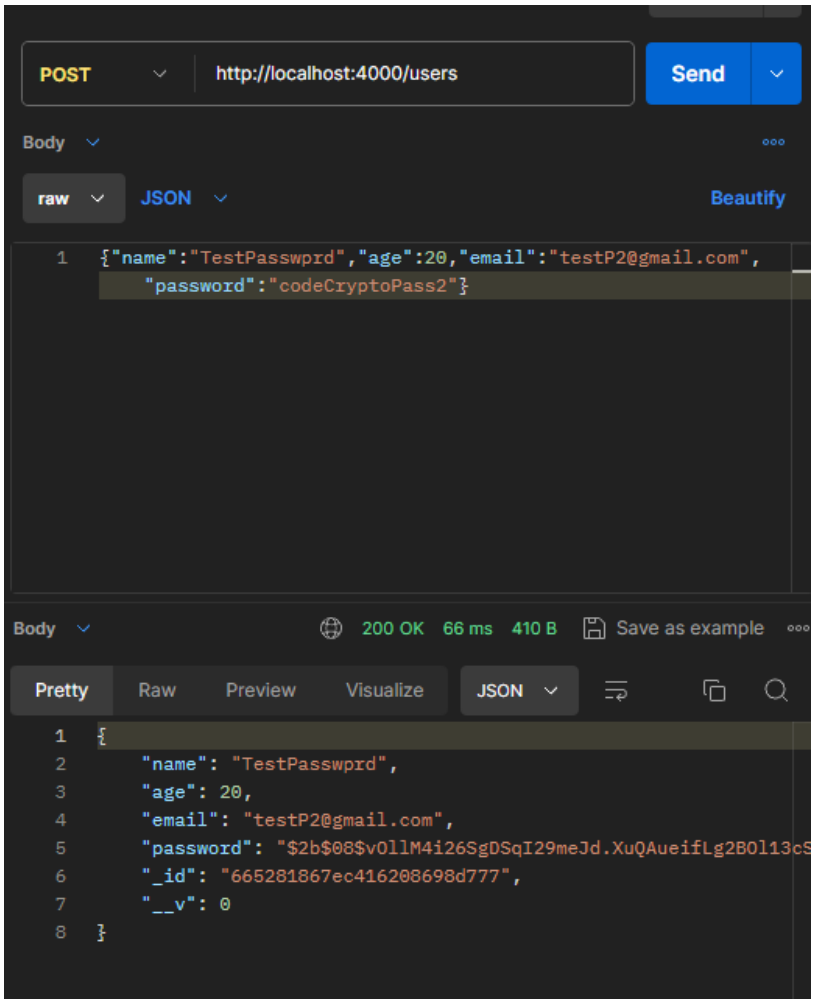
					ЖИТОМИРСЬКА ПОЛІТЕХНІКА.21.121.01.000 – Лр.4		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Алієв О.Є			Звіт з лабораторної роботи №4	Літ.	Арк. 1 / Аркушів
Перевір.		Сидорчук В.О					
Реценз.						ФІКТ, гр.ІПЗ-22-1	
Н. Контр.							
Зав.каф.							

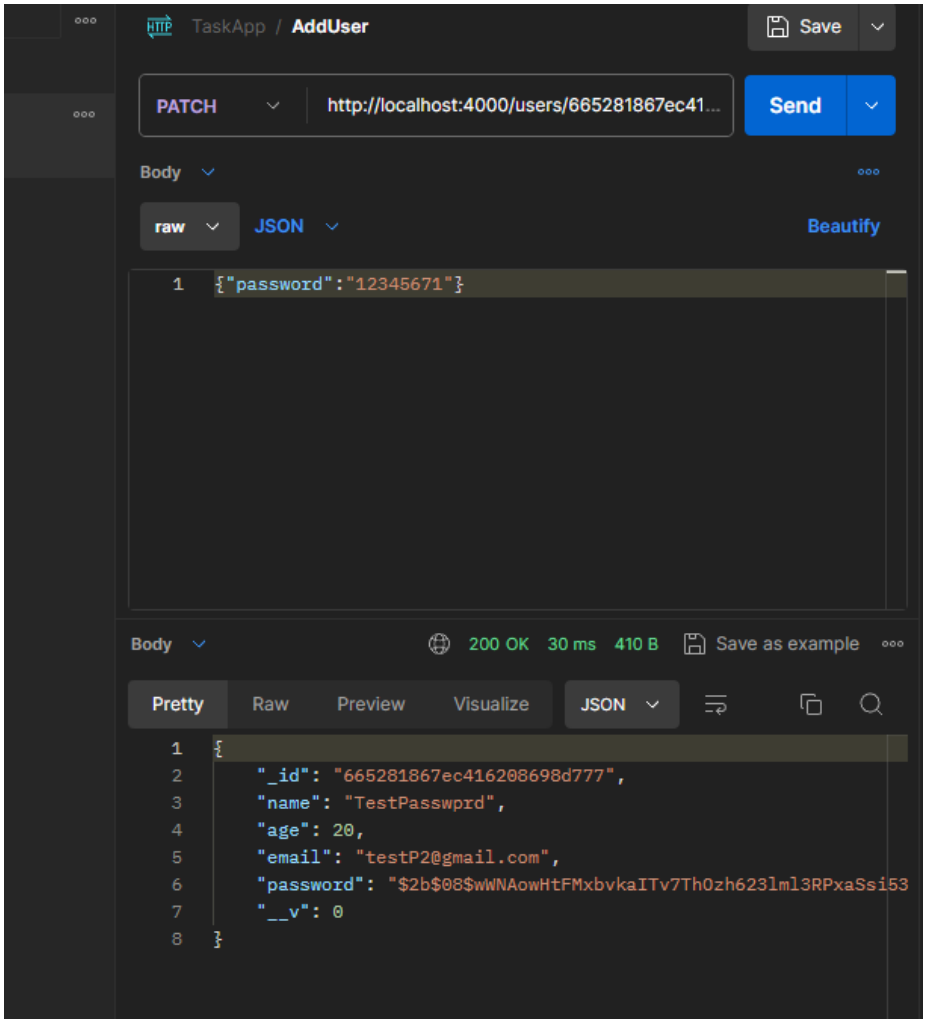
```
router.patch( path: '/users/:id', handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj
    try {
        const { id } = req.params
        const user : Query  = await User.findById(id)
        if (!user) {
            throw new Error("Not existing")
        }

        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)
    }
})
```

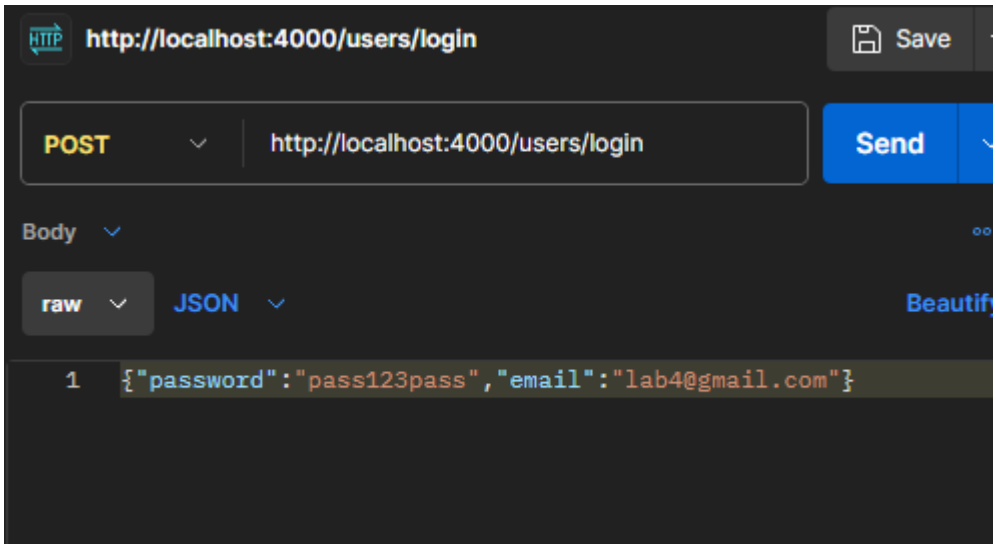
Переконайтесь в шифруванні пароля при відправці запитів на реєстрацію та на редагування





2. Створити запит /users/login на вхід.

Створіть запит /users/login



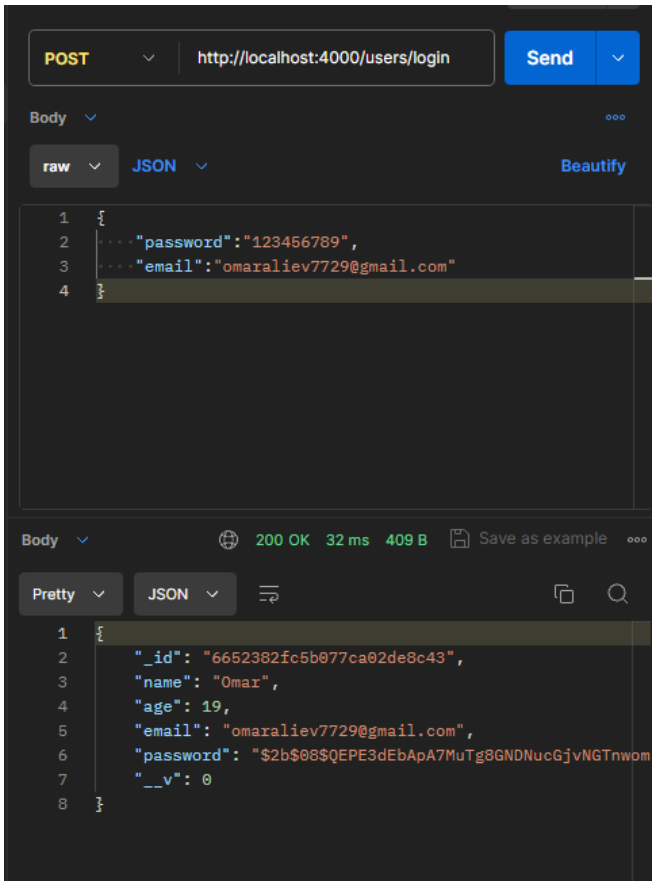
Створіть метод для аутентифікації

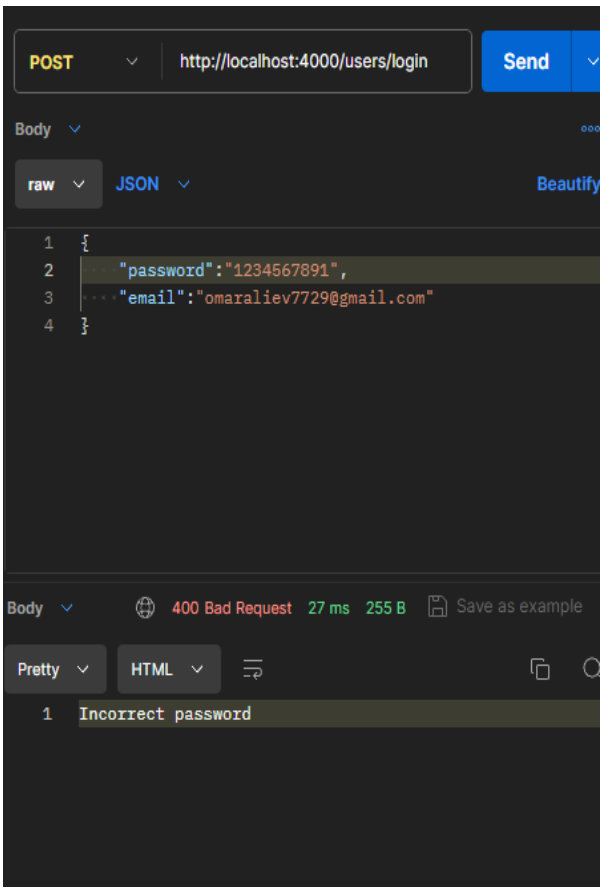
```
User.statics.findOneByCredentials = async (email, password) : Promise<any>  => {
    const user = await User.findOne({ email })
    if (!user) {
        throw new Error("Incorrect email")
    }
    const isMatch = await bcrypt.compare(password, user.password)
    if (!isMatch) {
        throw new 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 залогінимо 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
        }
    }]
})
```

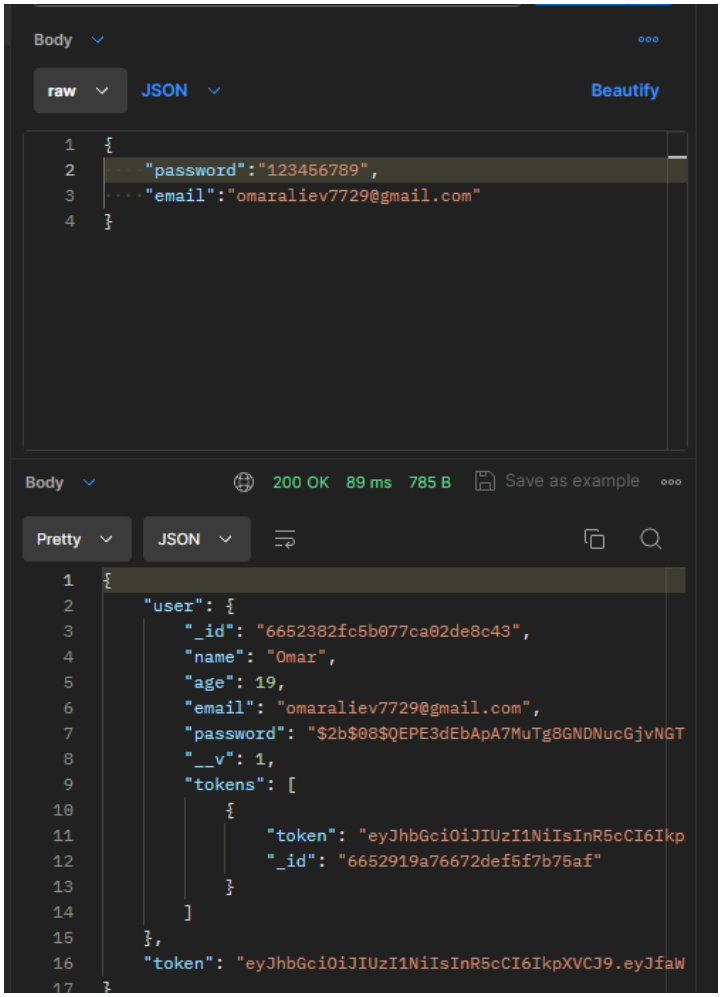
Створимо метод для генерації токена

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

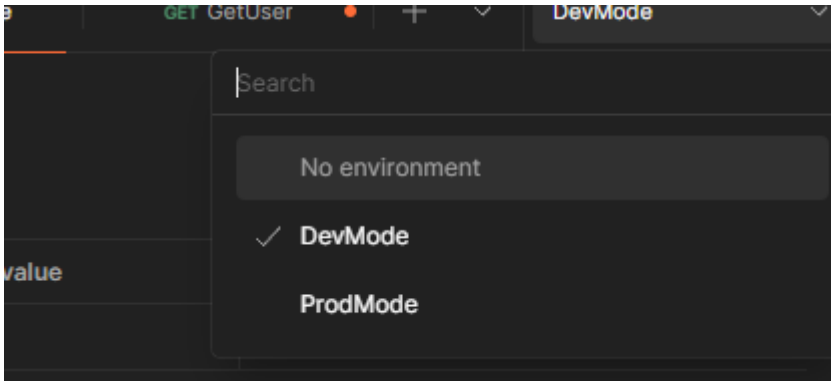
Забезпечимо генерацію токену при успішній аутентифікації

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

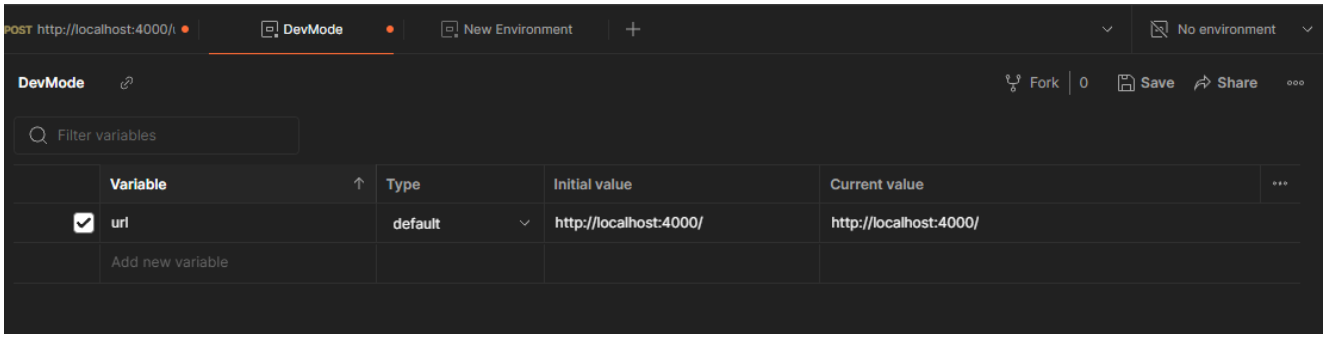
Тестуємо

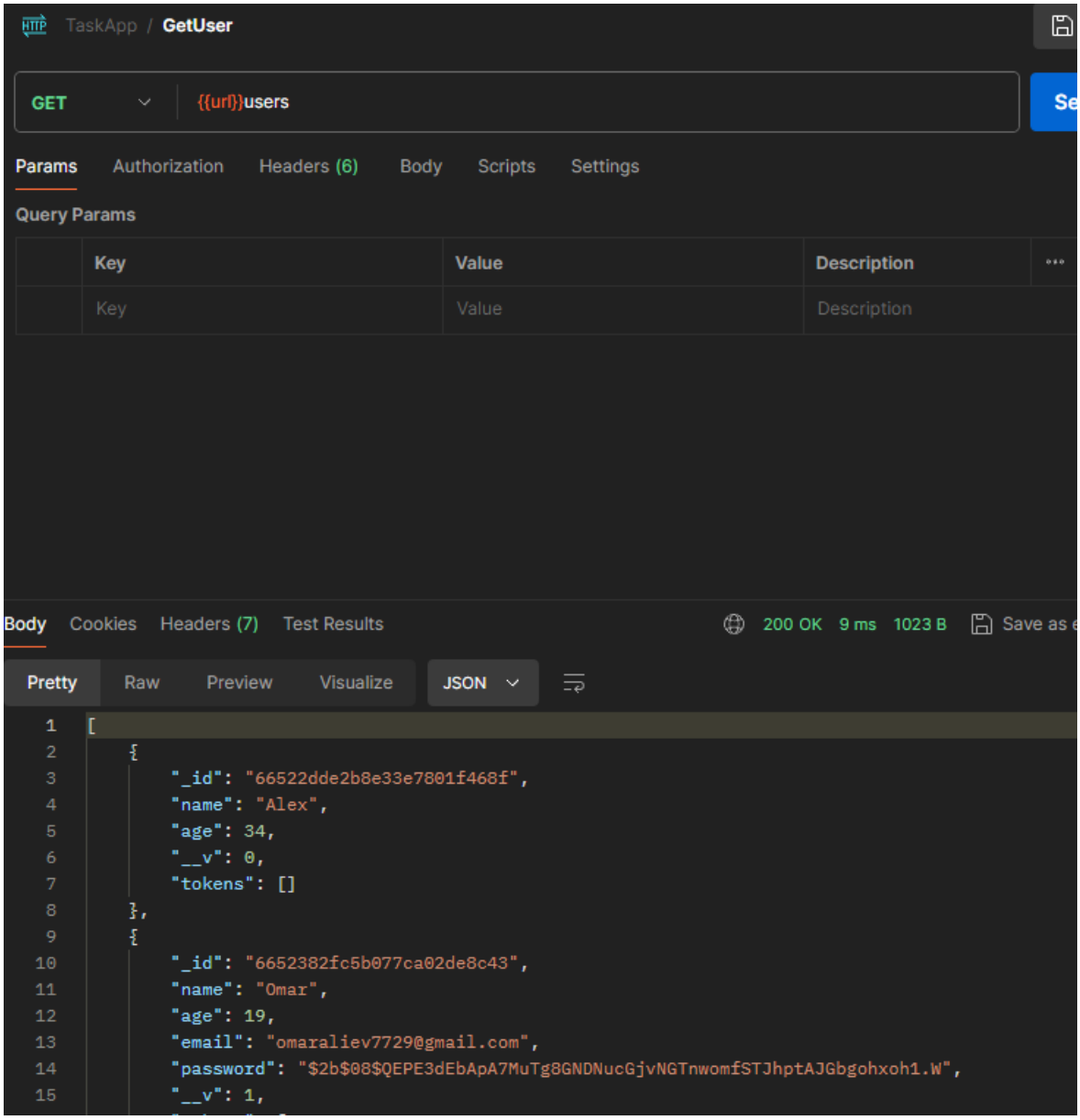


Середовище розробки і робоче середовище

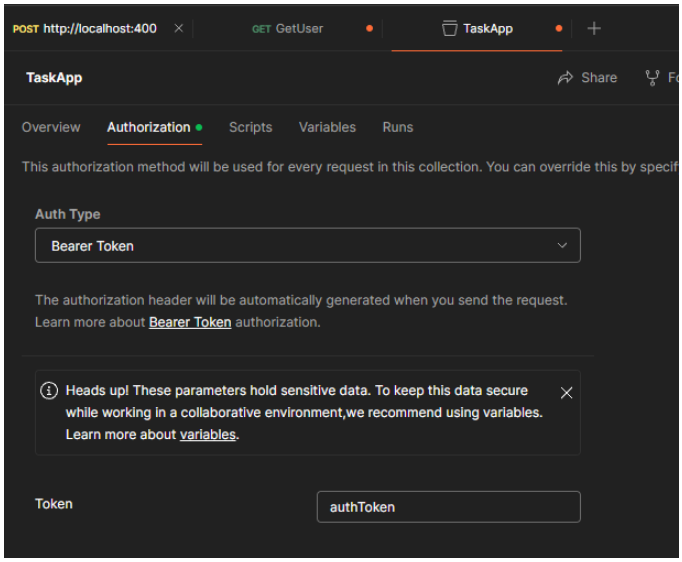


Змінна оточення *url*

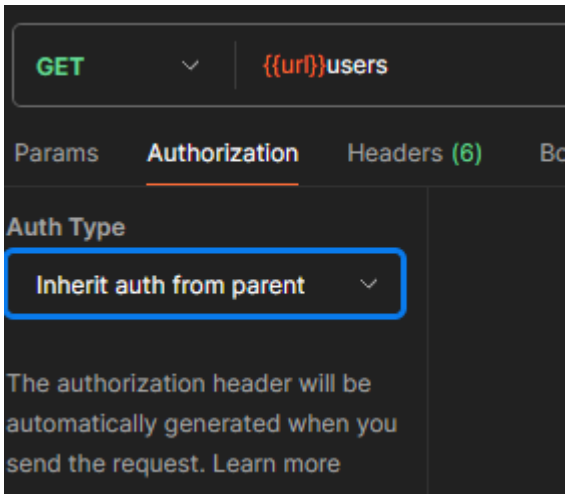




В Postman на рівні папки для наших запитів створити метод авторизації Bearer Token і в полі Token задати змінну оточення {{authToken}}



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

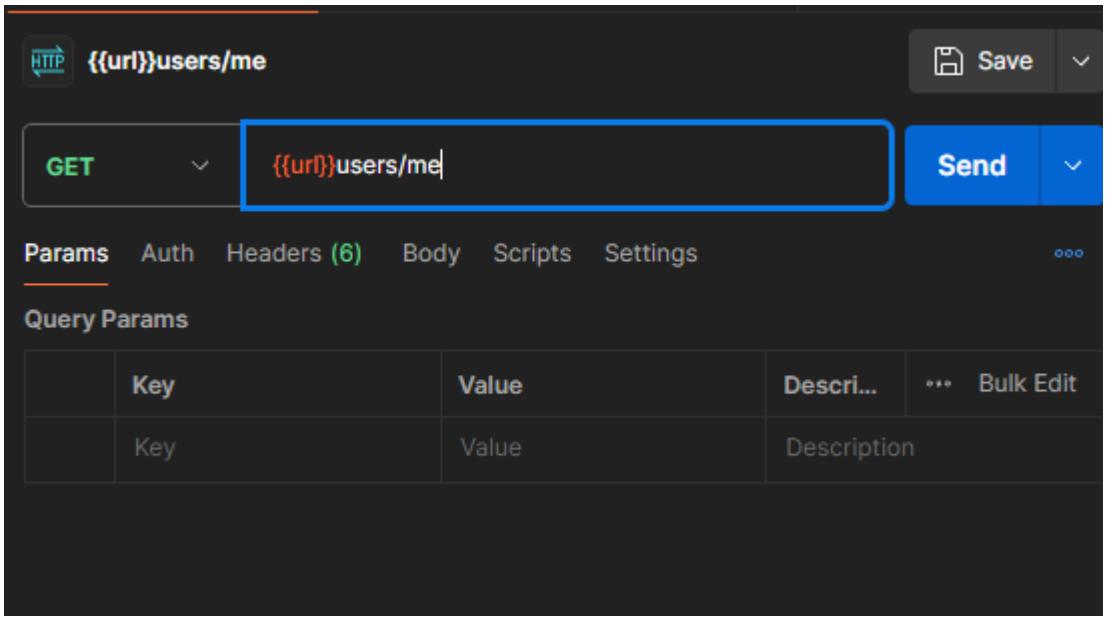


Автоматизуємо передачу значення змінній authToken

3. Примінити 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 токену і записує його в req.user

Створіть запит для перегляду даних про авторизованого користувача



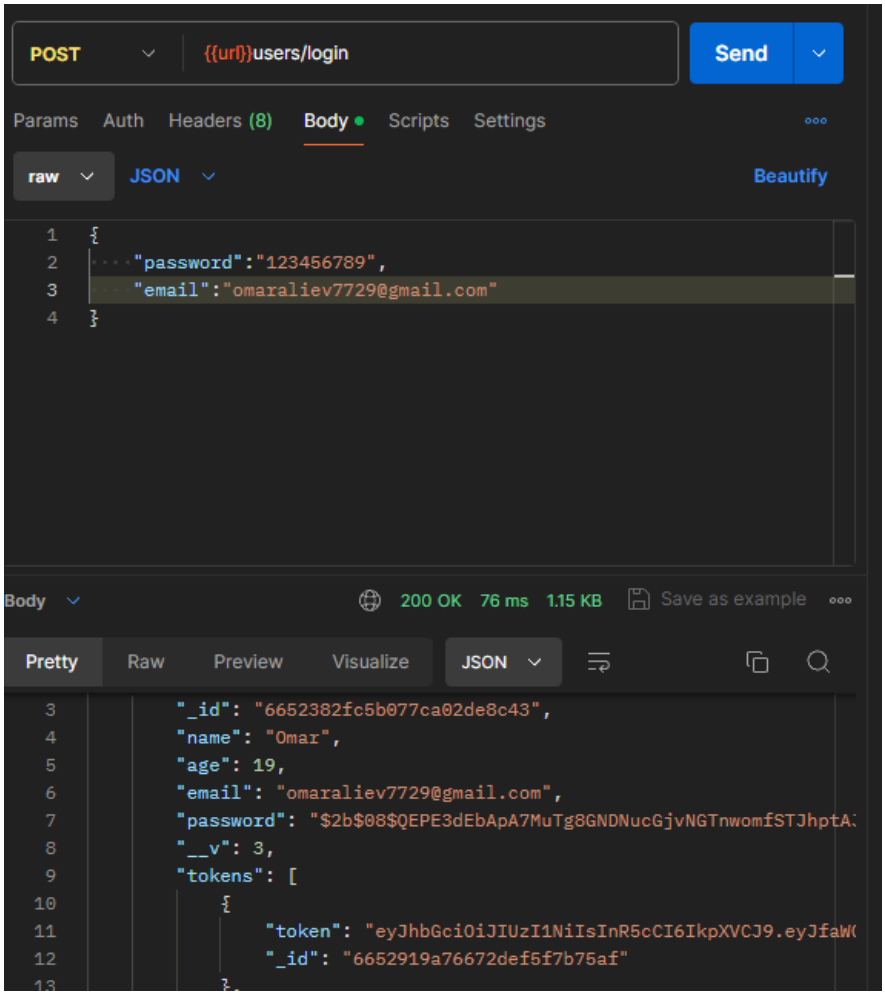
Здійсніть обробку даного запиту з попереднім виконанням middleware-функції auth

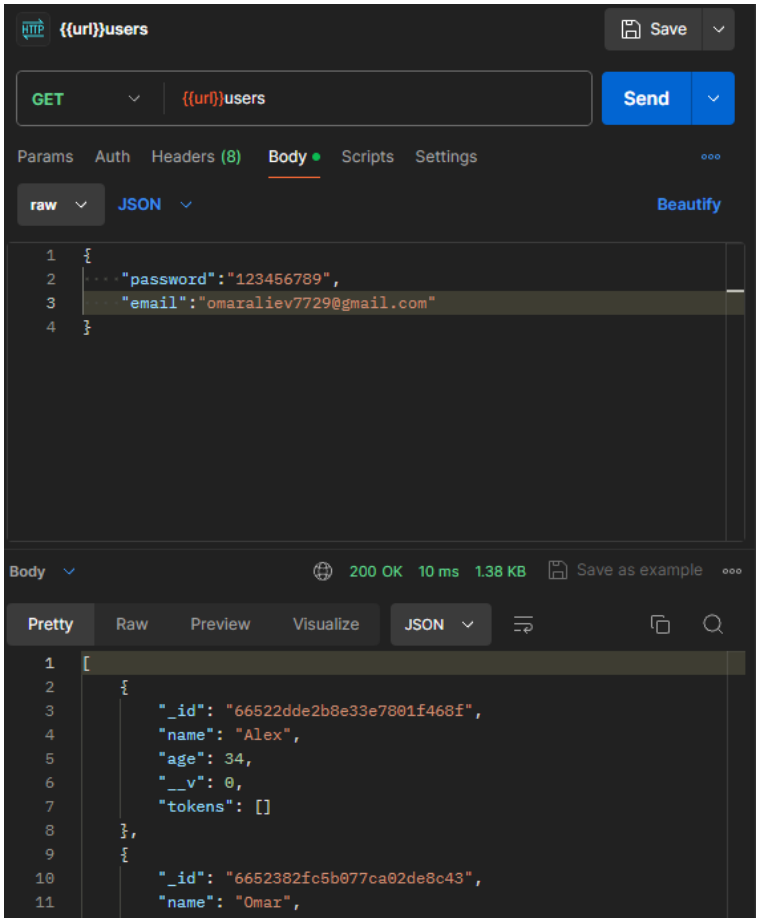
4. Створити запити /users/logout, /users/logoutAll. Вимагає авторизації. Видаляє запис про токен із БД.

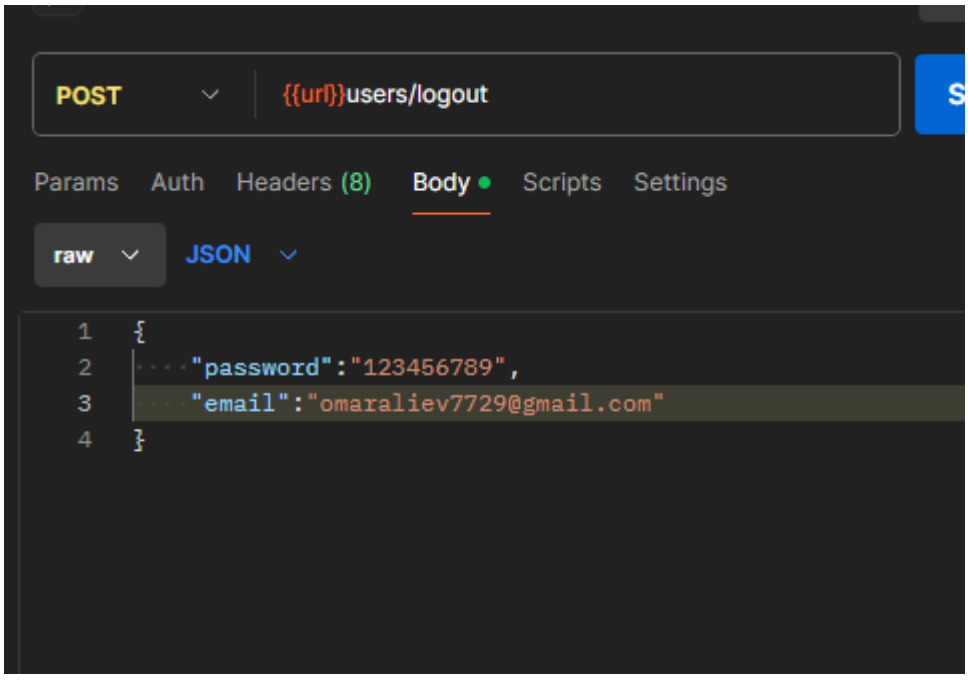
Створити POST-запит users/logout

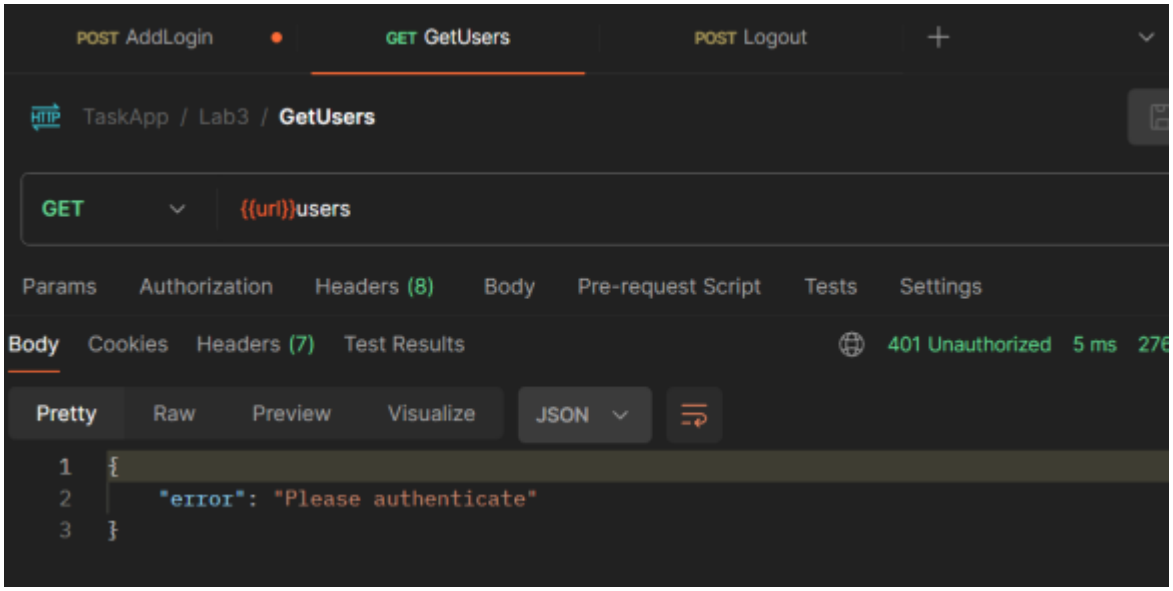
```
new *
router.post( path: "/users/logout", auth, async(req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , r
    try{
        req.user.tokens = req.user.tokens.filter((token) : boolean  => {
            return token.token !== req.token;
        })
        await req.user.save()
        res.send(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

Протестуйте виконання роботи




{{url}}users
Save
GET
{{url}}users
Send
Params
Auth
Headers (8)
Body
Scripts
Settings
raw
JSON
Beautify
{
"password":"123456789",
"email":"omaraliev7729@gmail.com"
}
Body
200 OK
10 ms
1.38 KB
Save as example
Pretty
Raw
Preview
Visualize
JSON
[
{
"_id": "66522dde2b8e33e7801f468f",
"name": "Alex",
"age": 34,
"__v": 0,
"tokens": []
},
{
"_id": "6652382fc5b077ca02de8c43",
"name": "Omar",


POST
{{url}}users/logout
Params
Auth
Headers (8)
Body
Scripts
Settings
raw
JSON
{
"password":"123456789",
"email":"omaraliev7729@gmail.com"
}



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