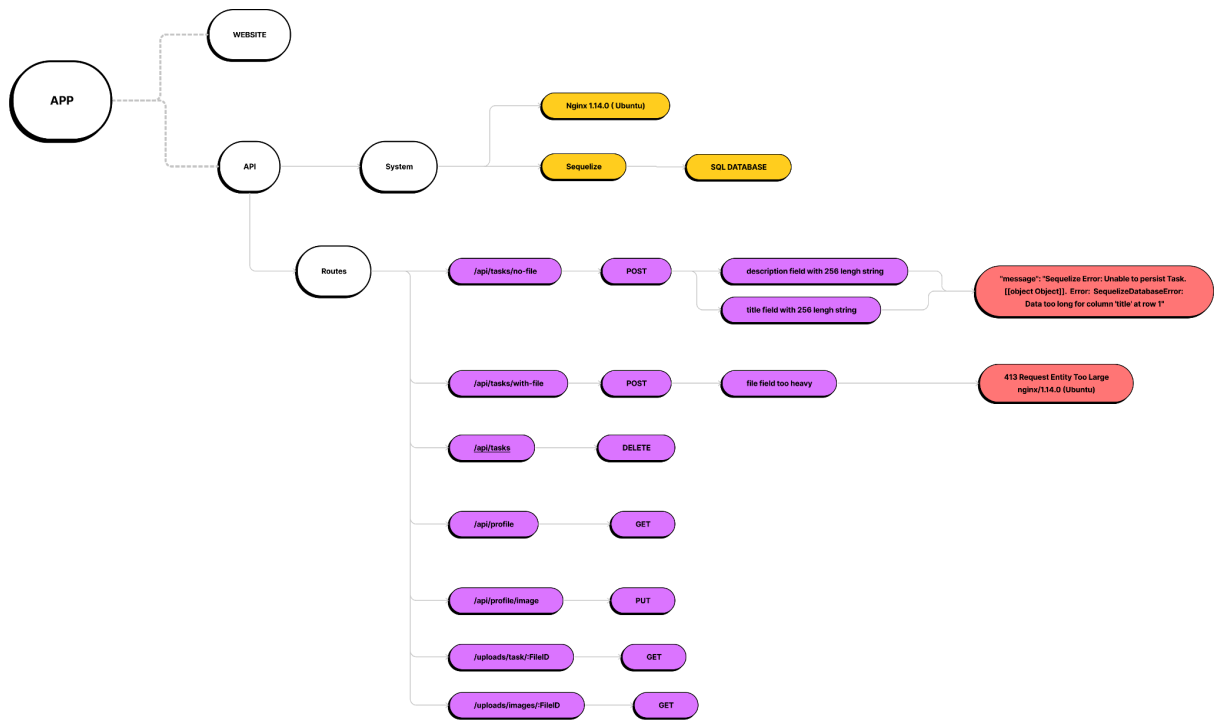


**M - Attack and defend your web applications**

## **Project Report**

antonin.alves-cardoso  
clement.chalopin  
djilani.cardineau  
louis.losson  
louis.vaubourdolle  
yohann.desravines

# ABOUT THE TARGET PROJECT



- Svelte
- Nginx 1.14.0 ubuntu
- server nodejs ?
- orm sequelize
- sql database

---

## PAGE ACCESS

With nothing to do these pages are accessible:

- <http://51.178.36.94:3000/login>
- <http://51.178.36.94:3000/profile>
- <http://51.178.36.94:3000/login.json>

---

## SECURITY ISSUES

<b>TYPE</b>	A03:2021-Injection
<b>VULNERABILITY</b>	The Front End only verifies if the token is stored, so we can write anything inside the local storage to bypass the verification.
<b>EXPLOIT</b>	See <b>LOCAL STORAGE EXPLOIT</b>
<b>FIX</b>	Create an API endpoint to verify the token and use it in the front router.

<b>TYPE</b>	A07:2021-Identification and Authentication Failures
<b>VULNERABILITY</b>	We can write a one character password
<b>EXPLOIT</b>	A lot of password will be enough weak to allow attacker to brute force the acces
<b>FIX</b>	Add a regex to verify the password before to create an account

<b>TYPE</b>	A09:2021-Security Logging and Monitoring Failures
<b>VULNERABILITY</b>	When you create a account with a email which has already been used we have a a http error from their api with this error message: Email is already in use
<b>EXPLOIT</b>	We can know if someone does have an account So we can create a list of users and brute force attack
<b>FIX</b>	Change the API result message to a less explicit message like “an error occured”

<b>TYPE</b>	A09:2021: Security Logging and Monitoring Failures
<b>VULNERABILITY</b>	In the upload file route when the title is up to 255 characters the backend route gives more information.
<b>EXPLOIT</b>	See <b>SEND LONG TITLE INPUT</b>
<b>FIX</b>	We need to catch this error from the database and not respond to the user.

<b>TYPE</b>	A01:2021-Broken Access Control
<b>VULNERABILITY</b>	We can't make files private
<b>EXPLOIT</b>	Anyone can access to all the file uploaded
<b>FIX</b>	Add a private feature to block access to other peoples

<b>TYPE</b>	A03:2021-Injection A04:2021-Insecure Design
<b>VULNERABILITY</b>	We can upload empty file with "": <i>base64</i> ," inside The API won't check the integrity of the file
<b>EXPLOIT</b>	We can upload anything we want
<b>FIX</b>	Check the file integrity in the API

<b>TYPE</b>	A03:2021-Injection A04:2021-Insecure Design
<b>VULNERABILITY</b>	We can upload any files
<b>EXPLOIT</b>	We can potentially upload script / binary , etc
<b>FIX</b>	Check the files integrity in the API

---

# HOW WE SECURED OUR APP

## 1 - File check

For the front part, we check the type of file thanks to the File interface of the typescript library which contains a type attribute. Which is created thanks to the ngx-file-drop lib which allows to upload files.

```
44     fileEntry.file((file: File) => {
45
46         if (!(file.type === 'image/png'
47             || file.type === 'application/pdf'
48             || file.type === 'application/vnd.openxmlformats-officedocument.wordprocessingml.document'
49             || file.type === 'image/jpeg')) {
50             this.error.showError("Only accept .pdf, .docx, .jpeg, .png")
51             return
52         }
53     })
```

## 2 - Jwt check for auth

We check the jwt token for all the auth routes

```
@file_upload.route('/upload', methods=['POST'])
@jwt_required()
@token_valid
@request_form
def upload(form, user):
    try:
        if not request.files.getlist("files"):
            return jsonify({'msg': 'File required'}), 404
        file = File(owner=user.id)
        if form.get('public').lower() == 'true':
            file.public = True
        file.name = request.files["files"].filename
        file.fs.put(request.files["files"])
        file.save()
        return jsonify({'msg': 'File successfully created'}), 201
    except ValidationError as error:
        return jsonify(error.to_dict()), 403
```



### 3 - We check the force of the user password

For prevent brute force attacks we force the user to have a strong password with Uppercase and special character.

```
@staticmethod
def password_strength():
    return r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*#?&])[A-Za-z\d@$!#%*?&]"
```

### 4 - We use restrictions on the models fields

```
class Users(Document):
    email = EmailField(unique=True, required=True, max_length=75)
    username = StringField(unique=True, required=True,
                           min_length=2, max_length=20)
    password = StringField(required=True, min_length=8, max_length=145)
    account = EmbeddedDocumentField(Users_account)
    profile_picture = StringField(required=False, max_length=300)
    created = DateTimeField(default=datetime.now)
    meta = {'db_alias': MONGO_DB_COMMON_ALIAS, 'collection': 'users'}
```

## 5 - Using middlewares to verify user permissions

We make some middleware for check the permissions of user when i try to access to protected routes.

```
6
7  def admin_required(func):
8      @wraps(func)
9      def decorator(*args, **kwargs):
10         identity = get_jwt_identity()
11
12         user = Users.objects(id=identity).first()
13         if not user.account.is_admin:
14             return admin_access_refused(), 401
15
16         return func(*args, **kwargs)
17     return decorator
18
19
20 def ban_checking(func):
21     @wraps(func)
22     def decorator(*args, **kwargs):
23         identity = get_jwt_identity()
24
25         user = Users.objects(id=identity).first()
26         if user.account.is_banned:
27             return user_banned(), 401
28
29         return func(*args, **kwargs)
30     return decorator
```

---

## 6 - Handle error outputs (using try / except)

For prevent Security Logging and Monitoring Failures we add try catch to not share sensitive information with the user.

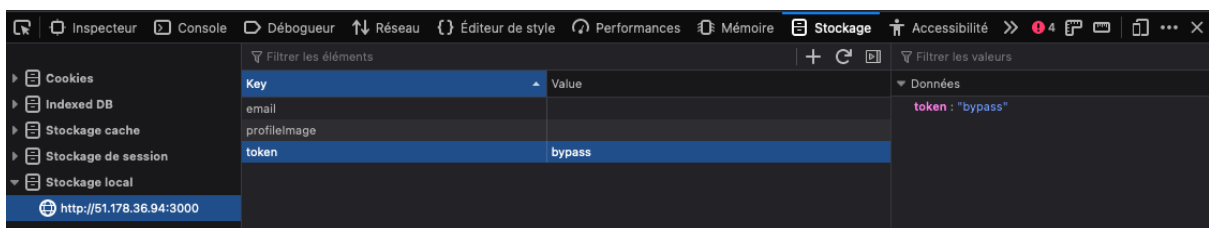
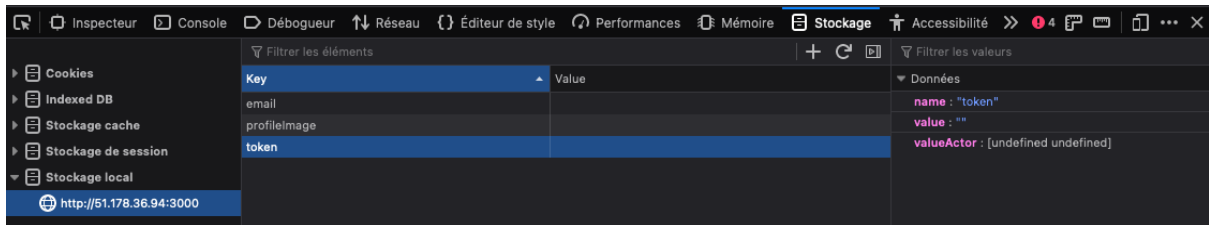
```
13 @account_delete.route('/', methods=['POST'])
14 @jwt_required()
15 @ban_checking
16 def me():
17     identity = get_jwt_identity()
18
19     try:
20         user = Users.objects(id=identity)
21         user.delete()
22         return jsonify({"message": "Account successfully deleted"}), 200
23     except:
24         return try_except_error(), 500
```

---

```
78
79 RESOURCE_NOT_FOUND = 'Resource not found'
80 RESOURCE_ALREADY_EXISTS = 'Resource already exists'
81 FRIEND_REQUEST_FAILED = 'Invalid friend request'
82 FRIEND_ACCEPT_FAILED = 'Invalid friend request accept'
83 FRIEND_DENY_FAILED = 'Invalid friend request deny'
84 FRIEND_DELETE_FAILED = 'Invalid friend request delete'
85 LOGIN_FAILED = 'Invalid email or password'
86 LOGIN_INVALID = 'Invalid resources provided, expect: email, password'
87 REGISTER_INVALID = 'Invalid resources provided, expect: email, username, password'
88 PASSWORD_INVALID = 'At least 8 chars with 1 uppercase, 1 lowercase, 1 number, 1 special'
89 POST_ADD_INVALID = 'Invalid resources provided, expect: content'
90 POST_ADD_FAILED = 'Invalid request: probably wrong image file given'
91 REQUEST_FORM_INVALID = 'Invalid request: content-type multipart/form-data expected'
92 REQUEST_JSON_INVALID = 'Invalid request: content-type application/json expected'
93 ALREADY_ADMIN = 'User has already admin access'
94 ADMIN_FAILED = 'Invalid request you must be admin'
95 BAN_ARGUMENT_INVALID = 'Invalid parameter [duration] - must be [0-9]*[d-h]'
96 UNLIKE_FAILED = 'Invalid unlike request'
97 MESSAGE_INVALID = 'Invalid resources provided, expect: content'
98 MESSAGE_FAILED = 'Invalid message request'
99 CHAT_FAILED = 'Invalid chat request'
```

# ANNEXES

## LOCAL STORAGE EXPLOIT



Now we have access to the Home page.



**We did not receive any patch from the other groups but for example we can improve the upload file part from our application.**

```
CONTENT_TYPE_ACCEPTED = ['image/jpeg', 'image/png', 'application/vnd.openxmlformats-officedocument.wordprocessingml.document', 'application/pdf']

@file_upload.route('/upload', methods=['POST'])
@jwt_required()
@token_valid
@request_form
def upload(form, user):
    try:
        if not request.files.getlist("files"):
            return jsonify({'msg': 'File required'}), 404
        file = File(owner=user.id)
        if form.get('public').lower() == 'true':
            file.public = True
        if not CONTENT_TYPE_ACCEPTED in request.files["files"].content_type:
            return jsonify({'msg': 'File type is not allowed'}), 404
        file.name = request.files["files"].filename
        file.fs.put(request.files["files"])
        file.save()
        return jsonify({'msg': 'File successfully created'}), 201
    except ValidationError as error:
        return jsonify(error.to_dict()), 403
```