RAPPORT DE STAGE

Léandro VEYRENC-CORDERO
12/05/2025 – 13/06/2025

Sommaire :

1. Présentation de l’entreprise :

Profluens est une S.A.S créé en mars 2023 pour mettre en avant la technologie Needle. Profluens a d’abord été présidé par Julien FALGAS, inventeur de needle et cofondateur de la société. Elle est désormais présidé par Antoine OBERLAENDER.

Needle est une plateforme collaborative innovante qui permet aux utilisateurs de découvrir des contenus de qualité sélectionnés par d'autres utilisateurs grâce à des connexions entre leurs contenus. Needle se distingue par son approche de "navigation contributive", qui favorise la curation et la mise en relation autour de contenus significatifs, tout en respectant l’attention des utilisateurs.

2. Problématique :

Pour le stage, je devais créer une serveur mail et une API REST. Sur la plateforme Needle, nous pouvons ajouter une annotation via une interface utilisateur. L’objectif de ce serveur mail et de l’API est d’ajouter une nouvelle façon d’ajouter ces annotations via un token.

Par exemple un utilisateur nommé John Dubois possède un compte Needle avec l’adresse mail john.dubois@gmail.com et associé au token A1B2C3D4. Il va donc pouvoir envoyer un mail à A1B2C3D4@in.needle.social avec comme sujet le lien qu’il souhaite ajouter et un message pour corps de message.

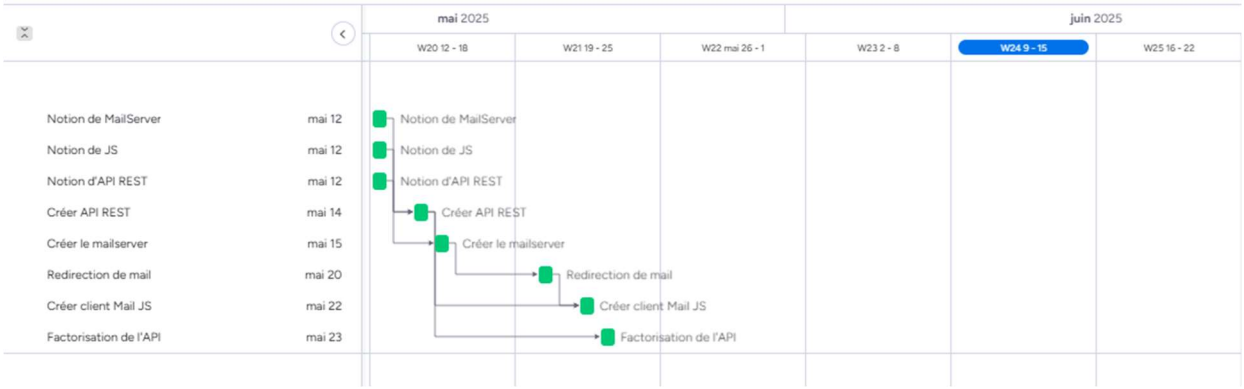
3. Contexte de travail :

Ce stage a eu lieu du 12 Mai 2025 au 13 Juin 2025. Il a été effectué en télétravail au Lycée Robert Schumann à Metz
Pour ce sujet, je me suis servi de Docker pour conteneuriser mes différentes applications et serveurs. J’ai aussi utilisé GitHub et Git afin de gérer, stocker et décrire les étapes de mon projet. Pour finir, j’ai utilisé Visual Studio Code comme IDE afin de développer mes applications.

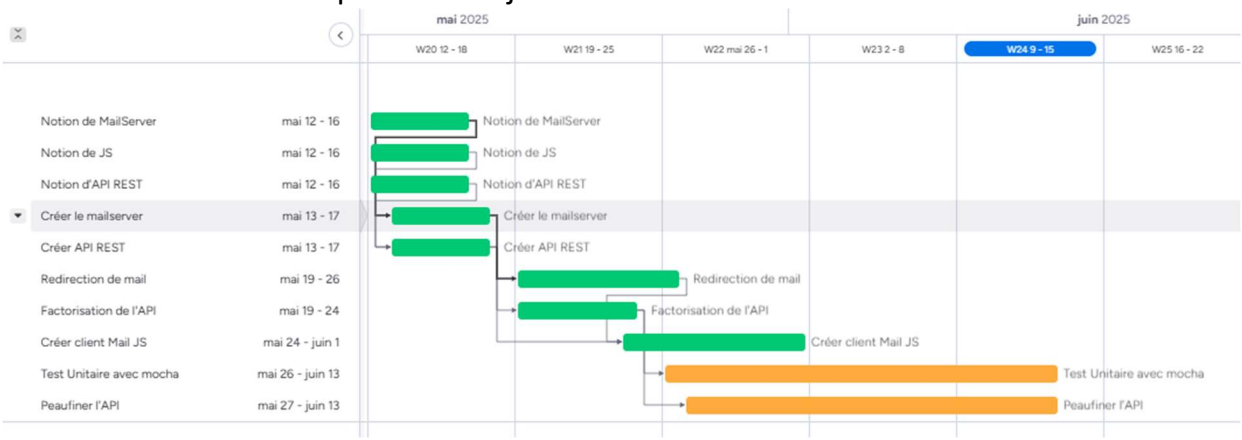
4. Gestion de projet :

Afin de détailler mon projet et clarifier mes tâches, j'ai utilisé un diagramme de GANTT qui me permet ainsi de poser des échéances et voir l'avancé du projet.

Voici 2 screenshots de mon diagramme de GANTT pour les échéanciers et les échéances :



Les échéances correspondent au jour où les tâches ont été terminées.

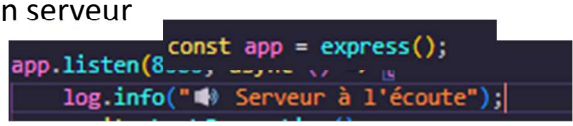


Les échéanciers correspondent au délai pour finir la tâche.

5. Étapes du projet :

Premièrement, je me suis renseigné et documenté sur les différentes technologies que j'allais utiliser tout au long de ce projet. J'ai donc d'abord regarder comment fonctionné un serveur de mail et plus particulièrement le conteneur docker mailserver. J'ai ensuite appris à développer en JavaScript et à comprendre ses objets JSON, le langage que j'ai utilisé pour développer l'API. Pour finir, je me suis renseigné sur ce qu'était une API et plus précisément une API REST.

Ensuite, j'ai commencé à développer mon API REST en JavaScript à l'aide d'un module nommé express. Ce module permet de créer un serveur web et donc notre API. Avec ces captures, notre serveur JavaScript écoute les requêtes sur le port



8080 de la machine qui l'héberge.

Notre API doit pouvoir recevoir des données afin de les traiter. J'ai donc créé un point de notre API. Ces points sont des adresses où peuvent être contacté l'API. Par exemple `app.post('/page/fil',` permet de poster des données à l'adresse local http://localhost:8080/page/fil et notre serveur va pouvoir les traiter.

Pour continuer, j'ai laissé de côté l'API pour commencer à configurer le serveur de mail. J'ai donc créer un fichier docker-compose pour démarrer le service et un fichier .env pour les variables environnement qui lui sont nécessaires au bon fonctionnement.
J'ai donc ensuite pu créer mes premiers utilisateurs afin de faire des tests et de mieux comprendre son fonctionnement.

Après, pour le système, je devais mettre en place des redirections puisque par exemple A1B2C3D4@in.needle.social n'existe pas mais est une adresse fictive. Pour mettre en place ces redirections, j'ai créé un compte final vers lequel les mails seront redirigé nommé token@in.needle.social. Pour créer des redirections, j'ai créé un fichier qui va contenir les règles comme celle-ci :

```
/^([a-zA-Z0-9]+)@in\.needle\.social\.localhost$/     token@in.needle.social.localhost
```

Cette règle redirige tous les mails sous la forme de <quelque chose>@in.needle.social.localhost vers token@in.needle.social.localhost. Pour activer les règles, nous devons ajouter ce fichier dans le fichier de configuration

```
virtual_mailbox_limit = 0
virtual_mailbox_maps = regexp:/etc/postfix/regex-aliases.cf
virtual_transport = lmtp:unix:/var/run/dovecot/lmtp
```

via virtual _mailbox_maps.

Après que la redirection et que la base de l'API a été développé, j'ai créé un client SMTP qui se connecte au compte token@in.needle.social pour lire les mails entrants extraire l'adresse de destination originale contenant le token, le sujet contenant le lien et le corps du message afin de les envoyer à l'API. Pour créer la connexion, j'ai créé un objet de la classe Imap avec un user (un mail), un mot de passe, un host et le port. Ces informations sont contenus dans le fichier .env.

```
const imapUser = new Imap({
  user: process.env.IMAP_USER,
  password: process.env.IMAP_PSW,
  host: process.env.IMAP_HOST,
  port: process.env.IMAP_PORT,
  tls: true,
  tlsOptions: { rejectUnauthorized: false }
});
```

Ensuite, avec cette connexion, nous allons pouvoir regarder les mails entrant sur la boite et en extraire les informations.

```
imap[0].on('mail', (numNewMsgs) => {
const seq = box.messages.total + ':' + box.messages.total;
const fetchImap = imap[0].seq.fetch(seq, {
    bodies: '',
    markSeen: true,
});

fetchImap.on('message', (msg) => {
    msg.on('body', (stream) => {
    simpleParser(stream, async (err, parsed) => {
        if (err) {
            log.error('Erreur de parsing :', err);
            return;
        }

        const to = parsed.to?.value?.[0]?.address || '';
        const from = parsed.from?.value?.[0]?.address || '';
        const subject = parsed.subject || '';
        const body = parsed.text || '';
        const date = parsed.date
        log.debug(parsed);
```

Ensuite on extrait le token depuis l'adresse mail grâce à une fonction qui sépare les chaîne de caractère avec le caractère @: `const tokens = extractTokens(to, imap[1]);`

Et pour finir, le client mail va envoyer les données à notre api :

```
const apiUrl = process.env.EXTERNAL_URL_API + "/page/fil";

try {
    let data = {
        tokenUser: tokens.tokenUser,
        from: from,
        subject: subject,
        bodyReq: body,
        date: date
    }
    const response = await fetch(apiUrl, {
                                    method: "POST",
                                    headers: {
                                        "Content-Type": "application/json",
                                        // 'Content-Type': 'application/x-www-form-urlencoded',
                                    },
                                    body: JSON.stringify(data)
                                    });

    if (response.status === 200) {
        log.info(data);
        log.info("✅ Données envoyées avec succès à l'API");
    } else {
        log.error(`❌ Erreur avec l'API : ${response.status}`);
    }
} catch (error) {
    log.error(` ! Erreur lors de l'envoi à l'API : ${error.message}`);
}
```

Notre API va traiter les données en utilisant des fonctions afin de séparer les tâches

et rendre plus maintenable le code :

```
app.post('/page/fil', async (req, res) => {
    const {from, tokenUser, bodyReq, subject, date} = req.body;
    console.log(req.body);

    if (await checkConnection()) {
        let sortiePostAnnotation;
        const sortieGetUser = await findUserWithToken(tokenUser);

        if (sortieGetUser.status !== 403 && sortieGetUser.status !== 401) {
            sortiePostAnnotation = await postAnnotation(sortieGetUser.laPersonne, from, sortieGetUser.urlId, date, bodyReq, subject);
        }


        let statusMail = await sendMailWithStatus(1, sortiePostAnnotation, from, subject, tokenUser)


        res.sendStatus(statusMail);
    } else {
        res.sendStatus(500);
    }
});
```

La fonction findUserWithToken nous permet de trouver un utilisateur grâce à un token en interrogeant l'API de la source de Needle. Elle nous retourne, le status de la réponse, l'objet de la personne et l'url du profil.

```
export async function findUserWithToken(token) {
    let status;
    let laPersonne = null;
    let userId = null;
    let urlGetId = null;
    let urlUserApiSource = null;
    try {
        log.debug('URL: ', getSourcesURL() + getApiEmailsTokenPath() + `/${token}`);

        const unePersonneRes = await fetch(getSourcesURL() + getApiEmailsTokenPath() + `/${token}`,
            {
                method: "GET",
                withCredentials: true,
                credentials: 'include',
                headers: {
                    "Authorization": 'Bearer ' + getApiTokenApiAccount(),
                    "Content-Type": "application/json"
                },
                redirect: 'manual'
            }
        );
        //const unePersonne = await unePersonneRes.json();
        log.debug(unePersonneRes);

        if (unePersonneRes.status === 303) {
            urlUserApiSource = unePersonneRes.headers.get('Location');
            userId = urlUserApiSource.split('/').pop();
            urlGetId = getSourcesURL() + getApiUserPath() + `/${userId}`;

            const laPersonneRes = await fetch(urlGetId,
                {
                    method: "GET",
                    withCredentials: true,
                    credentials: 'include',
                    headers: {
                        "Authorization": 'Bearer ' + getApiTokenApiAccount(),
                        "Content-Type": "application/json"
                    }
                }
            );
            laPersonne = await laPersonneRes.json();
            log.debug(laPersonne);
        }
    } catch (error) {
        status = 403;
        // if (error.response.status === 404) {
        //     status = 401;
        // }
        log.error(`❌ Erreur ${status} : ${error.message}`);
    }
    const sortie = {
        status: status,
        laPersonne: laPersonne,
        urlId: urlUserApiSource
    };
    log.debug("Sortie findUserWithToken : " + sortie);
    return sortie;

}
```

Ensuite la fonction postAnnotation, permet d'ajouter une annotation à l'API Source de Needle si l'expediteur du mail correspond au compte associé au token.

```
export async function postAnnotation(laPersonne, from, urlUserId, date, body, subject) {
    let statusPost;
    try {
        log.debug(laPersonne);
        if (laPersonne && laPersonne.email.toLowerCase() === from.toLowerCase()) {
            const uneAnnotation = getAnnotation(urlUserId, date, body, subject);
            const urlPostAnnotation = getSourcesURL() + getApiAnnotationPath() + `/${encodeURIComponent(urlUserId)}/${encodeURIComponent(subject)}`;
            log.debug("URL compte : " + urlUserId);
            log.debug("URL pour POST Annotation : " + urlPostAnnotation);
            log.debug("Annotation : " + uneAnnotation);
            const ajoutPage = await fetch(urlPostAnnotation,
                {
                    method: "POST",
                    withCredentials: true,
                    credentials: 'include',
                    redirect: 'manual',
                    headers: {
                        "Authorization": 'Bearer ' + getApiTokenApiAccount(),
                        "Content-Type": "application/json"
                    },
                    body: JSON.stringify(uneAnnotation)
                }
            );
            console.log(JSON.stringify(ajoutPage));
            statusPost = ajoutPage.status;
            log.info("Status fetch : " + statusPost);
        } else {
            statusPost = 401;
            log.error(`❌ Erreur ${statusPost} : Unauthorized Mauvais token`);
        }
    } catch (error) {
        statusPost = 403
        log.error(`❌ Erreur ${statusPost} : Forbidden : ${error}`);
    }
    return statusPost;
}
export function getAnnotation(urlId, date, body, subject) {
    const tbHashtags = [...body.matchAll(/#(\w+)/g)].map(match => match[1]); // match[1] => chaine sans le # ; match[0] => chaine avec le #
    log.debug(tbHashtags);
    const bodyAno = [
        {
            "type": "TextualBody",
            "value": body,
            "format": "text/plain",
            "purpose": "describing"
        }
    ];
    tbHashtags.forEach(tag => {
        bodyAno.push(
            {
                "type": "TextualBody",
                "value": tag ,
                "format": "text/plain",
                "purpose": "tagging"
            }
        );
    });
    const annotationVar = {
        "@context": "http://www.w3.org/ns/anno.jsonld",
        "type": "Annotation",
        "creator": urlId, // votre identifiant utilisateur ou URL
        "created": date, // date ISO du mail
        "body": bodyAno,
        "target": subject // cela pourrait être une URL ou un identifiant de ressource
    };
    log.debug(annotationVar);
    return {annotation: annotationVar};
}
```

Et pour finir, pour notifier l'utilisateur de l'ajout via nodemailer, un module pour envoyé des mails.

```
export async function sendMailWithStatus(type, status, from, subject, tokenUser) {
    // Type : fil ou patchwork => fil : 1 ; patchwork : 2
    if (type === 1) {
        if (status === 200) {
            const object = "Une nouvelle Fiche sur votre Fil";
            const message = `<h3>Bonjour ${from} ${tokenUser} !<h3>
                        <p>La page ${subject} a été correctement ajoutée à votre fil !</p>
                        <p><a href="https://needle.social">Needle.social</a></p>
                        <p>Ensemble, tissons un web plus humain</p>`;
            await envoyerEmailSMTP(from, object, message);
            log.info(`✅ Message 200 OK envoyé à ${from}`);
        } else if (status === 409) {
            const object = "Fiche déjà dans votre fil";
            const message = `<h3>Bonjour ${from} ${tokenUser} !<h3>
                        <p>La page ${subject} n'a pas pu être ajoutée à votre fil car elle y était déjà présente.</p>
                        <p><a href="https://needle.social">Needle.social</a></p>
                        <p>Ensemble, tissons un web plus humain</p>`;
            await envoyerEmailSMTP(from, object, message);
            log.warn(` !  Message 409 Conflict envoyé à ${from}`);
        } else if (status === 401) {
            const object = "Erreur Token";
            const message = `<h3>Bonjour ${from} !</h3>
                        <p>Nous n'avons pas pu effectuer votre demande puisque vos tokens ne correspondent pas à votre compte ni votre adresse email.</p>
                        <p><a href="https://needle.social">Needle.social</a></p>
                        <p>Ensemble, tissons un web plus humain</p>`;
            await envoyerEmailSMTP(from, object, message);
            log.error(`❌ Message 401 Unauthorized envoyé à ${from} `);
        } else if (status === 403) {
            const object = "Erreur dans l'ajout";
            const message = `<h3>Bonjour ${from} !</h3><br>
                        <p>Une erreur est survenue lors de l'ajout de votre page sur le fil. Veuillez réessayer ultérieurement.</p>
                        <p><a href="https://needle.social">Needle.social</a></p>
                        <p>Ensemble, tissons un web plus humain</p>`;
            await envoyerEmailSMTP(from, object, message);
            log.error(`❌ Message 403 Forbidden envoyé à ${from}`);
        }
```

```
async function envoyerEmailSMTP(destinataire, sujet, message) {
    log.debug({
            user: process.env.IMAP_REPLY_USER,
            pass: process.env.IMAP_REPLY_PSW
        });
    // Créer un transporteur SMTP
    const transporter = nodemailer.createTransport({
        host: process.env.IMAP_HOST,       // 🟦 Remplace par ton hôte SMTP
        port: 587,                      // ou 465 pour SSL
        secure: false,                  // true si port 465
        auth: {
            user: process.env.IMAP_REPLY_USER,
            pass: process.env.IMAP_REPLY_PSW
        },
        tls: {
            rejectUnauthorized: false  // Désactive la vérification SSL
        }
    });

    // Détails du message
    const mailOptions = {
        from: '"Needle Social" <reply@in.needle.social.localhost>',
        to: destinataire,
        subject: sujet,
        html: message
    };
    // Envoi
    try {
        const info = await transporter.sendMail(mailOptions);
        log.info('Email envoyé :', info.messageId);
    } catch (err) {
        log.error('Erreur lors de l'envoi :', err);
    }
}
```

Pour finir, pour tester chaque fonction de manière indépendante, j'ai utilisé mocha, qui permet de faire des tests unitaires avec la commande *npm test*.

Avec mocha, on peut faire en sorte qu'avant chaque test et après une tache peut

être réalisée, par exemple, ici avant on va définir les valeurs qu'a besoin l'API et créer un compte Mail de test. Et après chaque tâche, on supprime ce compte.

```
beforeEach(async function () {
    process.env.MAIL_DOCKER_NAME = "needle_mail_server-needle_mail_server-1";
    process.env.TARGETS_DOMAIN = "needle.social.localhost";
    setSourcesURL("http://localhost:3000");
    setSourcesRegisterToken("[redacted]");
    console.log(await checkAccount('test@in.needle.social.localhost'));
    if(!await checkAccount('test@in.needle.social.localhost')) {
        console.log("If checkAccount True");
        await createAccount('test', 'mdpTest');
    }
});
afterEach(async function () {
    await deleteAccount('test@in.needle.social.localhost');
});
```

6. Problèmes et solutions :

L'un des premiers problème qui est survenu lors des premiers test était le manque d'installation d'une dépendance, d'un module. Pour résoudre ce problème, il suffit juste de lancer dans un terminal la commande *npm install <nom_du_module>.*

Un autre problème qui pouvait survenir, c'est que lors du re-build des conteneurs docker, les fichiers de configuration étaient réinitialisés à l'état d'origine. Il a donc fallu créer moi-même les fichiers de configuration et de les monter en tant que volume dans le docker-compose. Le *:ro* à la fin d'une ligne indique à docker que c'est un fichier est non un dossier.

```
volumes:
  - ./server-mail/config:/tmp/docker-mailserver
  - ./server-mail/mail:/var/mail
  #- ./server-mail/postfix-spool:/var/spool/postfix
  - ./server-mail/ssl:/etc/ssl/mail
  #- ./server-mail/postfix/header_checks.pcre:/etc/postfix/header_checks.pcre:ro
  - ./server-mail/postfix/main.cf:/etc/postfix/main.cf:ro
  - ./server-mail/postfix/regex-aliases.cf:/etc/postfix/regex-aliases.cf:ro
```

Au début, les conteneur ne voulaient pas se construire puisque ils n'avaient pas les variables d'environnement. En effet, pour construire des conteneurs avec des variables d'environnement il faut exécuter la commande suivante :
docker compose --env-file .env up

Et pour finir, les derniers problèmes étaient simplement des fautes de frappe en oubliant des *s* ou en inversant deux lettres. C'est avec l'aide de l'IA que j'ai pu les résoudre sans perdre trop de temps.

7. Tests et vérifications :

Comme expliqué avant, mocha permet de faire des tests et de comparer des valeurs attendue.

Par exemple, une fonction pour vérifier la connexion avec les sources que l'on nomme *checkConnection()* est censé renvoyé *True* si la connexion est correctement établie.
Dans l'exemple ci-dessous, la variable *check* est la réponse retourné par la fonction. *Assert.equal* agit comme un opérateur. Ici, *check* doit être égal à *true* sinon le message *connection failed* sera affiché.

```
it('checkConnection', async () => {
    let check = await checkConnection();
    assert.equal(check, true,"connection failed");
});
```

8. Compétences :

- 1.1 Gérer le patrimoine informatique
-- 1.1.1 Recenser et identifier les ressources numériques
-- 1.1.2 Exploiter des référentiels, normes et standards adoptés par le prestataire informatique
-- 1.1.3 Mettre en place et vérifier les niveaux d'habilitation associés à un service
-- 1.1.4 Vérifier les conditions de la continuité d'un service informatique
-- 1.1.5 Gérer des sauvegardes
-- 1.1.6 Vérifier le respect des règles d'utilisation des ressources numériques

- 1.2 Répondre aux incidents et aux demandes d'assistance et d'évolution
-- 1.2.1 Collecter, suivre et orienter des demandes
-- 1.2.2 Traiter des demandes concernant les services réseau et système, applicatifs
-- 1.2.3 Traiter des demandes concernant les applications

- 1.3 Développer la présence en ligne de l'organisation
-- 1.3.1 Participer à la valorisation de l'image de l'organisation sur les médias numériques en tenant compte du cadre juridique et des enjeux économiques
-- 1.3.2 Référencer les services en ligne de l'organisation et mesurer leur visibilité
-- 1.3.3 Participer à l'évolution d'un site Web exploitant les données de l'organisation

- 1.4 Travailler en mode projet
-- 1.4.1 Analyser les objectifs et les modalités d'organisation d'un projet
-- 1.4.2 Planifier les activités
-- 1.4.3 Évaluer les indicateurs de suivi d'un projet et analyser les écarts

- 1.5 Mettre à disposition des utilisateurs un service informatique

-- 1.5.1 Réaliser les tests d'intégration et d'acceptation d'un service
-- 1.5.2 Déployer un service
-- 1.5.3 Accompagner les utilisateurs dans la mise en place d'un service

- 1.6 Organiser son développement professionnel
-- 1.6.1 Mettre en place son environnement d'apprentissage personnel
-- 1.6.2 Mettre en œuvre des outils et stratégies de veille informationnelle
-- 1.6.3 Gérer son identité professionnelle
-- 1.6.4 Développer son projet professionnel