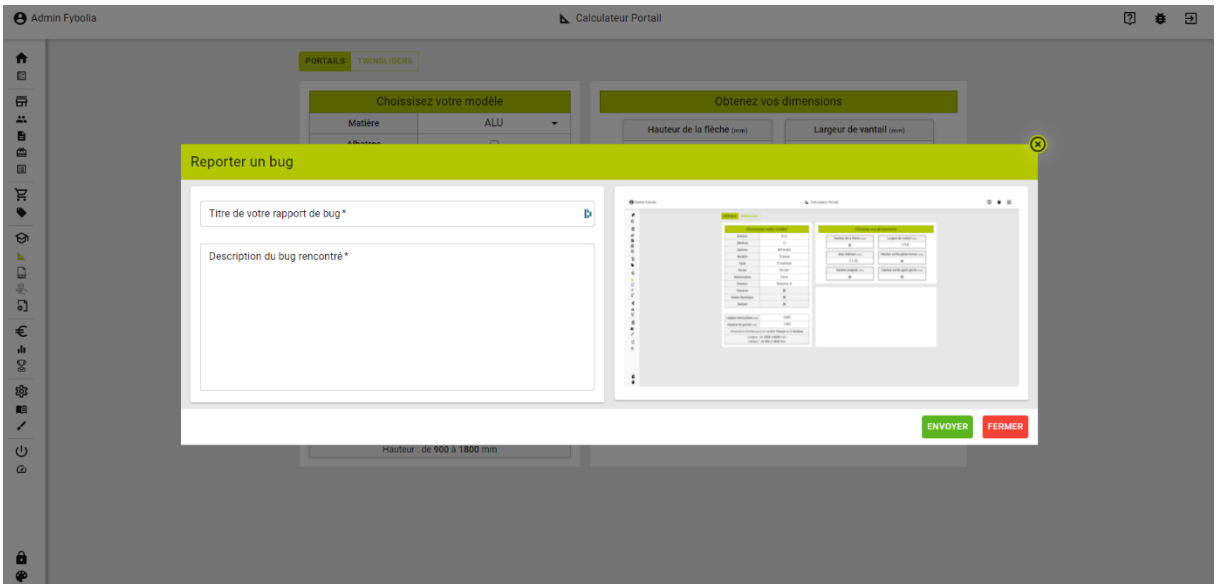
C2.1 : Collecter, suivre et orienter des demandes

Pendant mon stage, après avoir réalisé ma mission principale (création d'une application de calculs de côtes de portail), j'ai été missionné sur le « bug report » de l'ERP en développement de l'entreprise.

C'est un espace pour que les salariés de l'entreprise puissent faire des retours aux développeurs sur les bugs qu'ils ont puis rencontrés pendant leur session. Ce module existait déjà avant mon arrivé, mais sous une forme différente.

En effet, le module prenait un « screenshot » de la page où l'utilisateur avait appuyé sur le bouton « bug report » et permettait à l'utilisateur de rentrer le titre de sa demande et une bref description. Ensuite, ces informations étaient transférées par mail à l'ensemble du service développement.



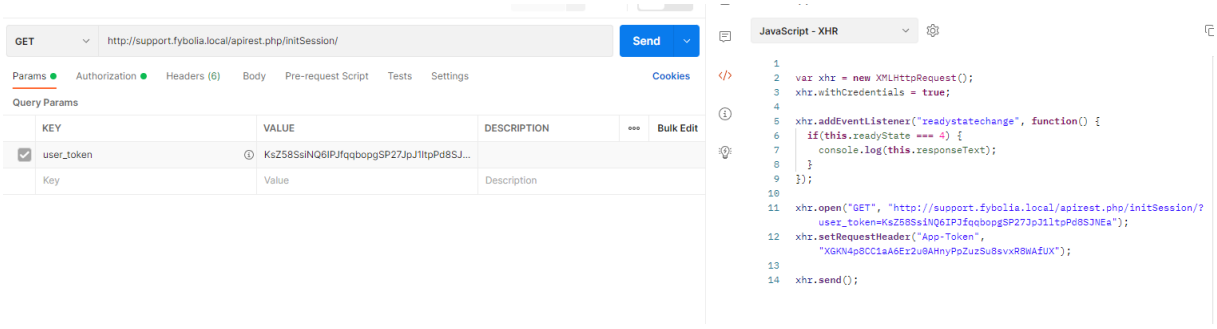
Le « bug report » de l'ERP

Cela faisait doublon, car l'entreprise était dotée de GLPI (un gestionnaire de parc informatique libre) dont il se servait aussi pour la création de tickets à destination des techniciens. Le service n'avait pas encore eu le temps de relier le bug report de l'ERP à GLPI.

Grâce à la documentation de GLPI, j'ai pu voir qu'il avait mis en place une API REST pour gérer ce genre de module. J'ai donc pu me servir de POSTMAN, pour tester les requêtes API dont j'avais besoin.

Pour commencer, j'ai dû demander à l'administrateur réseau de l'entreprise de me créer un compte super-admin GLPI pour l'intranet (Le système API de GLPI demande deux tokens pour la réalisation des requêtes).

J'ai commencé par créer la première requête afin d'obtenir le token de session :



Requête API pour l'obtention du « session token »

Après avoir réalisé l'ensemble de mes requêtes sur POSTMAN pour être sûr que tout fonctionne parfaitement (poster un ticket, poster une image et lié les deux ensembles), j'ai pu réaliser la partie back-end pour l'intranet.

J'ai pu garder l'ensemble des routes de l'ancien système, mais j'ai dû remplacer la fonction qui envoyait un mail à l'équipe, elle ne serait plus utile, car c'est GLPI qui devait gérer cela (en fonction du technicien à qui s'adressait le ticket).

J'ai commencé par coder la requête pour le token de session :

```
async function getSessionTokken() {
    let resultat = undefined;
    try {
        await new Promise((resolve, reject) => {
            let xhr = new XMLHttpRequest()
            let timeout_xhr = setTimeout(() => reject(), 200);
            xhr.open('GET', url + '/initSession/', true)
            xhr.setRequestHeader('X-Requested-With', 'xmlhttprequest')
            xhr.setRequestHeader('App-Token', appToken)
            xhr.setRequestHeader('Authorization', userToken)
            xhr.onreadystatechange = function () {
                if (xhr.readyState === 4) {
                    if (xhr.status === 200) {
                        let response = JSON.parse(xhr.responseText)
                        resultat = response.session_token;
                        clearTimeout(timeout_xhr);
                        resolve()
                    } else {
                        clearTimeout(timeout_xhr);
                        reject()
                    }
                }
            }
            xhr.send()
        });
    } catch (err) {
        logger.error(err);
    } finally {
        return resultat;
    }
}
```

Puis une fonction qui, grâce au nom LDAP (*Lightweight Directory Access Protocol*) présent sur l'intranet, devait retrouver le nom du salarié sur GLPI :

```
async function getIdByName(sessionToken = '', usernameGLPI = '') {
    let idRequester = undefined;
    try {
        if (sessionToken !== undefined && sessionToken !== '' && usernameGLPI !== undefi
ned && usernameGLPI !== '') {
            await new Promise((resolve, reject) => {
                var options = {
                    'method': 'GET',
                    'url': url + '/search/User?forcedisplay[0]=2&forcedisplay[1]=5&crite
ria[1][field]=1&criteria[1][searchtype]=2&criteria[1][value]=' + usernameGLPI,
                    'headers': {
                        'App-Token': appToken,
                        'Session-Token': sessionToken
                    }
                };
                request(options, function (error, response) {
                    if (error) {
                        logger.error(error);
                        reject()
                    }
                    let res = (JSON.parse(response.body)).data[0];
                    idRequester = (Object.values(res))[1]
                    resolve();
                });
            });
        }
    } catch (err) {
        logger.error(err)
    }
    return idRequester
}
```

Je devais ensuite créer le ticket GLPI avec les informations de l'utilisateur récoltées sur le module bug report de l'intranet (nom de la page, description, url et le demandeur) :

```
async function createGLPITicket(sessionToken = '', ticket_infos = { titre: '', url_page:
'', nom_page: '', description: '', nom_demandeur: '' }, id_demandeur) {
    let create_ticket = undefined;
    if (sessionToken !== '') {
        try {
            await new Promise((resolve, reject) => {
                let xhr = new XMLHttpRequest();
                let data = {
                    input: {
                        name: ticket_infos.titre.trim(),
                        status: 2,
                        content: 'Nom de la page : ' + ticket_infos.nom_page.trim() +
'\n\rDescription : ' + ticket_infos.description.trim() + ' \n\rURL : ' + ticket_infos.ur
l_page.trim() + ' \n\rDemandeur: ' + ticket_infos.nom_demandeur.trim(),
                        urgency: 3,
                        impact: 3,
                        priority: 3,
                        _groups_id_assign: 3,
                        _users_id_requester: id_demandeur,
                        itilcategories_id: 14,
                        requesttypes_id: 7,
                        // _users_id_assign: []
                    }
                }
```

```
                xhr.open('POST', url + '/ticket/', true)
                xhr.setRequestHeader('X-Requested-With', 'xmlhttprequest')
                xhr.setRequestHeader('Content-Type', 'application/json')
                xhr.setRequestHeader('App-Token', appToken)
                xhr.setRequestHeader('Session-Token', sessionToken)
                let timeout_xhr = setTimeout(() => reject(), 200);
                xhr.onreadystatechange = function () {
                    if (xhr.readyState === 4) {
                        if (xhr.status === 200 || xhr.status === 201) {
                            let response = JSON.parse(xhr.responseText)
                            clearTimeout(timeout_xhr);
                            create_ticket = response;
                            resolve()
                        } else {
                            clearTimeout(timeout_xhr);
                            reject()
                        }
                    }
                }
                xhr.send(JSON.stringify(data))
            });
        } catch (err) {
            logger.error(err);
        }
    }
    return create_ticket
}
```

Je devais par la suite attacher la pièce-jointe (le screenshot) au ticket en fonction de l'id du ticket que nous venions de créer :

```
async function addAttachmentToTicket(ticket_id = '', img_data64 = '', sessionToken = '')
{
    let uploaded = false;
    const filename = moment().utcOffset(0, true).format('x') + '.jpeg';
    const path_img = "temp/" + filename;
    try {
        if (
            ticket_id !== undefined && ticket_id !== '' && img_data64 !== undefined && i
mg_data64 !== '' && sessionToken !== undefined && sessionToken !== ''
        ) {
            var base64Data = img_data64.replace(/^data:image\/jpeg;base64,/, "");
            fs.writeFileSync(path_img, base64Data, 'base64');

            await new Promise((resolve, reject) => {
                var options = {
                    'method': 'POST',
                    'url': url + '/Document',
                    'headers': {
                        'App-Token': appToken,
                        'Session-Token': sessionToken
                    },
                    formData: {
                        'uploadManifest': JSON.stringify({
                            input: {
                                name: filename,
                                items_id: ticket_id + '',
                                itemtype: "Ticket"
                            }
                        }),
                        'file': {
                            'value': fs.createReadStream(path_img),
                            'options': {
                                'filename': path_img,
                                'contentType': null
                            }
                        }
                    }
                };
```

Enfin, je devais « killer » la session générée par la première requête :

```
async function killSession(sessionToken = '') {
    let sessionKilled = false;
    try {
        if (sessionToken !== undefined && sessionToken !== '') {
            await new Promise((resolve, reject) => {
                var options = {
                    'method': 'GET',
                    'url': url + '/killSession',
                    'headers': {
                        'App-Token': appToken,
                        'Session-Token': sessionToken
                    }
                };
                request(options, function (error, response) {
                    if (error) {
                        logger.error(error);
                        reject()
                    }
                    sessionKilled = true;
                    resolve();
                });
            });
        }
    } catch (err) {
        logger.error(err)
    }
    return sessionKilled
}
```

Voilà la fonction qui effectue toutes ses actions :

```
export const sendBugReportMail = async (req, res, next) => {

    try {
        if (!req.user) {
            return next(new ErrorResponse("Une erreur s'est produite.", 500));
        }
        if (Utils.strUndef(req.body.titre) && Utils.strUndef(req.body.description) && Ut
ils.strUndef(req.body.nom_page) && Utils.strUndef(req.body.url_page)) {
            const sessionToken = await getSessionTokken()
            const idRequester = await getIdByName(sessionToken, req.user.sAMAccountName)
            const new_ticket = await createGLPITicket(sessionToken, {
                titre: req.body.titre,
                url_page: req.body.url_page,
                nom_page: req.body.nom_page,
                description: req.body.description,
                nom_demandeur: req.user.sAMAccountName
            }, idRequester);
            let attach_ok = false;
            if (new_ticket !== undefined) {
                attach_ok = await addAttachmentToTicket(new_ticket.id, req.body.data_uri
_image, sessionToken)
            }
            killSession(sessionToken)
            if (attach_ok) {
                res.status(200).json({ status: "ok" });
            } else {
                res.status(200).json({ status: "nok" });
            }
        } else {
            res.status(200).json({ status: "nok" });
            return next(new ErrorResponse("Informations manquantes", 500));
        }
    } catch (err) {
        logger.error(err);
        return next(new ErrorResponse("Une erreur s'est produite.", 500));
    }

}
```

Voilà ce que cela pouvait donner dans GLPI :

ID	Titre	Statut	Dernière modification	Date d'ouverture	Priorité	Demandeur - Demandeur	Attribué à - Technicien	Catégorie	Date d'échéance
1	PC Ultra lent	Nouveau	2022-04-16 11:05	2022-04-16 11:05	Haute	Dupond Jean		Problèmes > Matériel	
ID	Titre	Statut	Dernière modification	Date d'ouverture	Priorité	Demandeur - Demandeur	Attribué à - Technicien	Catégorie	Date d'échéance

Ceci est une représentation, je n'ai pas eu le droit de récupérer un vrai ticket