PROJET WEB D-UNITY

Le réseau social des développeurs DWWM

Giovan HAMADAINE

29/11/2023

DWWM Via Formation

Sommaire

INTRODUCTION

Ma passion pour le développement informatique est ancrée en moi depuis ma plus tendre enfance. Même si mon cursus initial m'a orienté vers des domaines éloignés de l'univers numérique, ma soif d'apprendre ne m'a jamais quitté. Profitant de chaque moment libre, je me suis plongé dans des ouvrages, des cours en ligne et des tutoriels liés à l'informatique. Cet apprentissage autodidacte m'a permis non seulement de maîtriser plusieurs langages de programmation, mais aussi de rester constamment à jour avec les évolutions rapides du secteur technologique.

Face à une lassitude croissante vis-à-vis de mon domaine d'activité précédent, j'ai ressenti le besoin d'embrasser pleinement ma passion et de la transformer en compétence professionnelle reconnue. Après avoir exploré différentes voies, mon choix s'est porté sur l'école Via Formation.

La culmination de cette formation a été la mise en œuvre du projet de fin d'études. Durant un mois intensif, j'ai eu l'opportunité de conceptualiser et de développer le projet D-UNITY. Ce projet m'a offert un terrain d'expérimentation idéal pour mettre en pratique mes compétences en matière de développement front-end et back-end. J'ai utilisé des technologies de pointe, largement plébiscitées dans le monde professionnel. Mon objectif principal était de maîtriser les bonnes pratiques, d'apprendre à organiser un projet de grande envergure et de comprendre chaque étape de la réalisation d'une application, de la conception à la mise en production.

Au final, la réalisation du projet D-UNITY a été pour moi bien plus qu'un simple exercice académique. Elle m'a permis d'affiner mes compétences, de renforcer ma confiance en moi et de me préparer efficacement à intégrer le monde professionnel du développement informatique

LISTE DES COMPÉTENCES COUVERTES PAR LE PROJET

I. **Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité**
   A. Maquetter une application
   B. Réaliser une interface utilisateur web statique et adaptable
   C. Développer une interface utilisateur web dynamique

II. **Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité**
   A. Créer une base de données
   B. Développer les composants d'accès aux données
   C. Développer la partie back-end d'une application web ou web mobile

RÉSUMÉ DU PROJET

Dans le vaste univers du numérique, il existe un besoin croissant pour les développeurs de se connecter, d'échanger et de collaborer en temps réel. En réponse à cette demande, j’ai créé D-UNITY, une plateforme spécialement conçue pour la communauté des développeurs. Plus qu'un simple réseau social, D-UNITY allie simplicité, efficacité et un fil d'actualité fluide dédié aux sujets techniques.

Dès leur première connexion, les utilisateurs, qu'ils soient débutants ou experts, sont plongés dans un espace où la technologie est reine. Ils peuvent partager des astuces, des bouts de code, réagir aux publications, et bien sûr, moduler leurs propres contributions. La spécificité de D-UNITY ? Chaque membre a la liberté, et est même encouragé, à contribuer à l'amélioration et à l'évolution de la plateforme. Ma vision est de faire de D-UNITY un projet communautaire où chaque développeur peut apporter sa pierre à l'édifice.

Grâce à une mise à jour en temps réel du fil d'actualité, chaque interaction est immédiatement visible par la communauté. Que ce soit une nouvelle astuce partagée, une question posée ou une amélioration suggérée pour le site, tout est reflété instantanément.

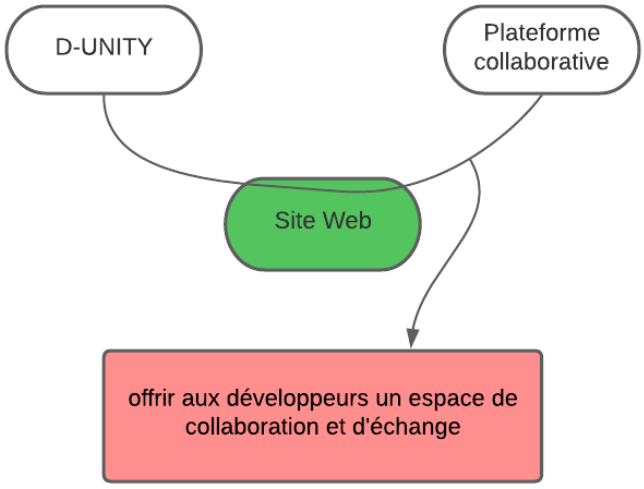
En conclusion, D-UNITY n'est pas simplement un réseau social : c'est une plateforme collaborative, conçue par des développeurs, pour des développeurs, et en constante évolution grâce à la participation active de sa communauté. Pour l'instant, je lance D-UNITY, impatient de voir où cette aventure collaborative me mènera.

CAHIER DES CHARGES

I. Besoins et Objectifs de l'application

A. Besoins

D-UNITY naît d'un besoin croissant d'offrir aux développeurs un espace de collaboration et d'échange en temps réel. Dans un monde où la programmation et le développement deviennent de plus en plus collaboratifs, il est essentiel de disposer d'une plateforme qui permet non seulement d'échanger des idées, mais aussi de voir les contributions des autres en temps réel. De plus, avec l'essor des applications mobiles et web, le besoin de modularité et de réutilisabilité du back-end est devenu un impératif, sans oublier l'importance cruciale de la sécurité pour protéger les données des utilisateurs.



B. Objectifs

L'objectif principal de D-UNITY est de créer une passerelle de connectivité entre les développeurs, qu'ils soient novices ou experts. En favorisant la collaboration, je vise à enrichir la communauté de connaissances partagées et d'expertise. D-UNITY se veut

également évolutif, capable de s'adapter et de croître en fonction des besoins changeants de sa communauté. Enfin, l'intuitivité est au cœur de ma démarche, garantissant une expérience utilisateur fluide et sans heurts, indépendamment du niveau technique de l'utilisateur.

C. Cibles

D-UNITY s'adresse à un large éventail d'utilisateurs. Les développeurs juniors y trouveront un terrain fertile pour apprendre et collaborer, tandis que les développeurs seniors pourront partager leur expertise tout en bénéficiant des perspectives fraîches des novices. Les entreprises pourront également profiter de D-UNITY pour repérer des talents ou pour collaborer sur des projets spécifiques. Enfin, les communautés de développeurs y trouveront un espace idéal pour organiser des événements ou des ateliers.

Marc Bernard - 47 ans
Le responsable RH

Marc travaille dans une entreprise de technologie qui est en constante quête de nouveaux talents. Il utilise D-UNITY pour identifier des développeurs prometteurs et établir des partenariats avec des freelancers pour des projets à court terme.

Léa Fontaine - 29 ans
L'organisatrice de com

Léa est à la tête d'une communauté de développeurs dans une grande ville. Elle organise régulièrement des hackathons et des ateliers de codage. Léa voit en D-UNITY la plateforme parfaite pour rassembler les membres de sa communauté, promouvoir des événements et stimuler la collaboration entre ses membres.

Sam Leroy - 35 ans
La Développeuse sénoir

Une développeuse logiciel avec plus de dix ans d'expérience. Elle maîtrise plusieurs langages de programmation et est reconnue pour son expertise en architecture logicielle. Samantha aime partager son savoir-faire avec les jeunes développeurs et cherche des moyens de rester connectée aux dernières tendances du développement.

Alexis Dupont - 22 ans
Le Développeur Junior

vient de terminer ses études en informatique. Passionné par le développement web, il est toujours à la recherche de plateformes pour apprendre et mettre en pratique ses compétences en codage. Alexis est particulièrement intéressé par le développement front-end et souhaite contribuer à des projets open source pour enrichir son portfolio.

II. Users Stories

Ce tableau de "user stories" est un outil de planification qui sert à décrire les fonctionnalités d'un site ou d'une application du point de vue de l'utilisateur final. Chaque ligne représente une histoire d'utilisateur (user story) qui suit un modèle standard : "En tant que [type d'utilisateur], je souhaite [action ou fonctionnalité], afin de [objectif ou bénéfice]."

(Voir annexe 2)

En tant que	Je souhaite	Afin de
Visiteur	consulter le fil d'actualité	voir les dernières publications et les échanges entre développeurs
Visiteur	accéder à la page des trends	voir les publications les plus populaires
Visiteur	consulter les profils	connaître les membres de la communauté, sans accéder aux informations privées
Visiteur	m'inscrire sur la plateforme	rejoindre la communauté et participer aux discussions
Visiteur	me connecter sur la plateforme	accéder à mon profil et interagir avec la communauté
Développeur	publier des messages dans le fil d'actualité	partager des idées, poser des questions et interagir avec les autres membres
Développeur	réagir aux publications des autres	liker, commenter et participer aux discussions
Développeur	personnaliser mon profil	mettre à jour mes informations et montrer mon historique d’interactions
Développeur	naviguer entre différentes sections de l'application	accéder aux fonctionnalités et aux informations pertinentes
Modérateur	surveiller les publications et les commentaires	assurer que le contenu reste respectueux et conforme aux directives
Modérateur	intervenir en modifiant ou supprimant les contenus inappropriés	maintenir la qualité des échanges sur la plateforme

Modérateur	avoir un profil similaire à celui du développeur	permettant d'interagir avec la communauté tout en gardant certaines prérogatives de modération
Administrateur	gérer les comptes des utilisateurs	créer, modifier, supprimer des comptes et définir leurs rôles
Administrateur	avoir un contrôle total sur la page d'accueil et les publications	supprimer n’importe quel message si nécessaire
Administrateur	intervenir sur les profils	apporter des modifications aux informations des utilisateurs si nécessaire

III. MVP

D-UNITY est conçu comme une plateforme dynamique centrée sur la collaboration et l'échange entre développeurs. Au cœur de cette application, trois rôles se démarquent : le développeur, le modérateur et l'administrateur.

Chaque utilisateur, quel que soit son rôle, s'authentifie grâce à un système sécurisé basé sur un email et un mot de passe. Une fois connecté, l'accès aux conditions générales d'utilisation et à la politique de confidentialité est facilité, garantissant une transparence totale vis-à-vis de nos membres.

A. Le Développeur

Le développeur, en se connectant, découvre un fil d'actualité interactif. Il peut y publier des messages, réagir aux publications des autres et naviguer entre différentes sections de l'application. Chaque développeur possède un espace personnel, un profil où il peut non seulement consulter ses informations et l'historique de ses interactions mais aussi les modifier selon ses besoins.

B. Le Modérateur

Le rôle du modérateur, quant à lui, est crucial pour le bon fonctionnement de D-UNITY. En plus de posséder un profil similaire à celui du développeur, le modérateur a la lourde tâche de veiller à la qualité des échanges sur la plateforme. Il surveille les publications et les commentaires, s'assurant que le contenu reste respectueux et conforme à nos directives. En cas de dérive, il peut intervenir en modifiant ou en supprimant les contenus inappropriés.

C. L’administrateur

L'administrateur, pilier de la plateforme, gère les comptes des utilisateurs. Il a la capacité non seulement de créer, de modifier ou de supprimer des comptes, mais aussi de définir leurs rôles, assurant ainsi une hiérarchie et une organisation fluide au sein de D-UNITY. Comme tous les autres utilisateurs, l'administrateur a également son propre espace profil, lui permettant de gérer ses informations personnelles.

IV. Fonctionnalités détaillées des pages

D-UNITY est une plateforme collaborative dédiée aux développeurs. Son architecture a été soigneusement élaborée pour offrir une expérience utilisateur optimale, adaptée à chaque type d'utilisateur, qu'il soit simple visiteur, membre connecté, modérateur ou administrateur.

(Voir annexe 1)

A. La page d'accueil (fil d’actualité)

Sur la page d'accueil (fil d'actualité), un visiteur a la possibilité de consulter le fil d'actualité. Un membre connecté, outre la consultation, peut publier, "liker", commenter et supprimer ses propres messages. Le modérateur, tout en ayant les mêmes droits que le membre, peut en outre modérer les commentaires. L'administrateur a le contrôle total, pouvant même supprimer n'importe quel message.

B. La page Trend (Publication les plus populaires)

La page des trends offre aux visiteurs la chance de consulter les publications les plus populaires. Les membres connectés peuvent interagir avec ces publications en les "likant" et en y laissant des commentaires. Les modérateurs peuvent intervenir pour modérer ces commentaires si nécessaire, tandis que l'administrateur possède une liberté totale d'action.

C. Le Profil

Quant à la page de profil, les visiteurs peuvent consulter les profils sans toutefois accéder à des détails privés. Les membres connectés ont la possibilité de personnaliser leurs profils, suivre d'autres membres et retracer leurs interactions passées. Les modérateurs ont une vision plus complète des profils, et les administrateurs peuvent intervenir pour apporter des modifications si nécessaire.

D. L’inscription

La page d'inscription est principalement destinée aux visiteurs qui souhaitent rejoindre la communauté. Ils peuvent y remplir et soumettre le formulaire d'inscription.

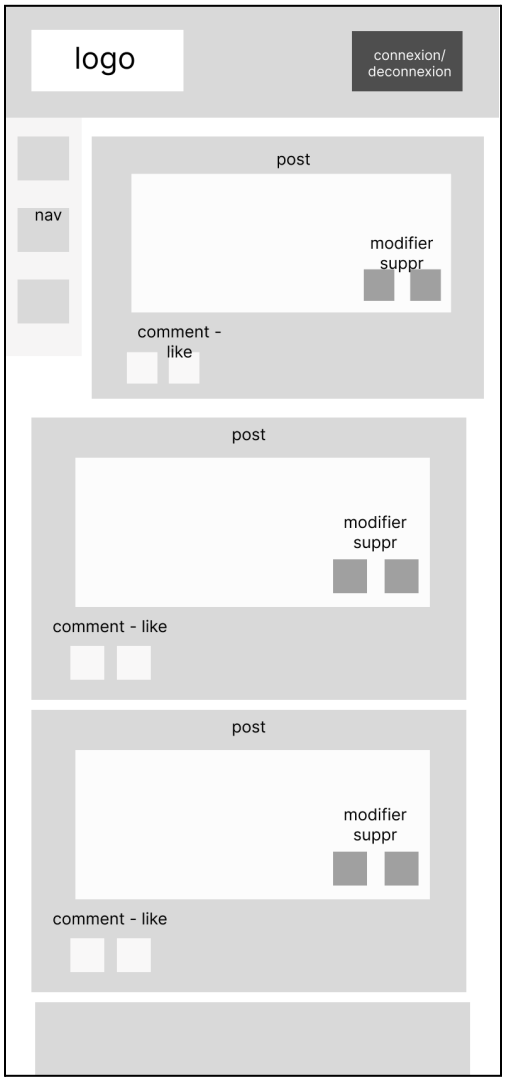
Toutefois, les membres déjà connectés, qu'ils soient modérateurs ou administrateurs, sont directement redirigés vers la page d'accueil s'ils tentent d'accéder à cette page.

E. La connexion

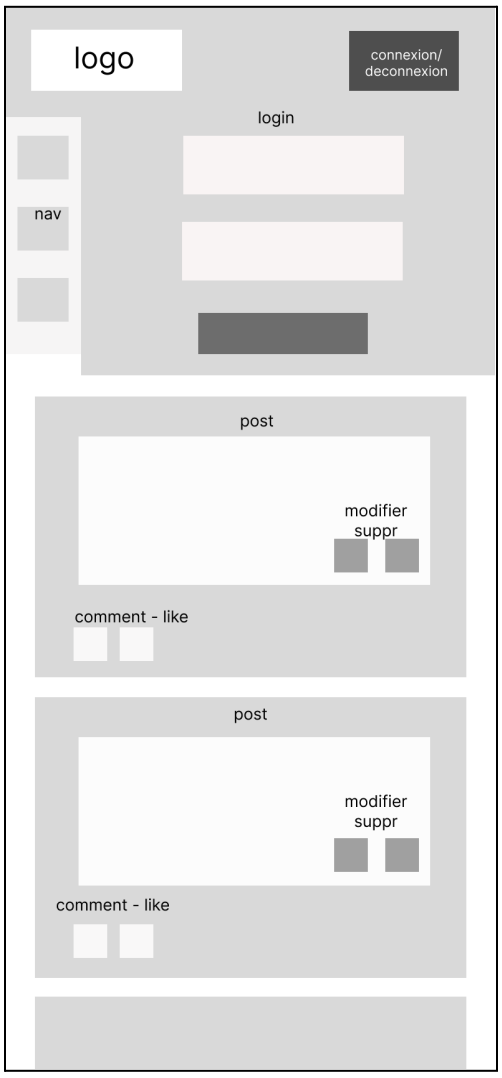
Enfin, la page de connexion permet aux visiteurs de se connecter à leurs comptes respectifs. Cependant, tout comme pour la page d'inscription, les membres déjà connectés sont redirigés vers la page d'accueil.

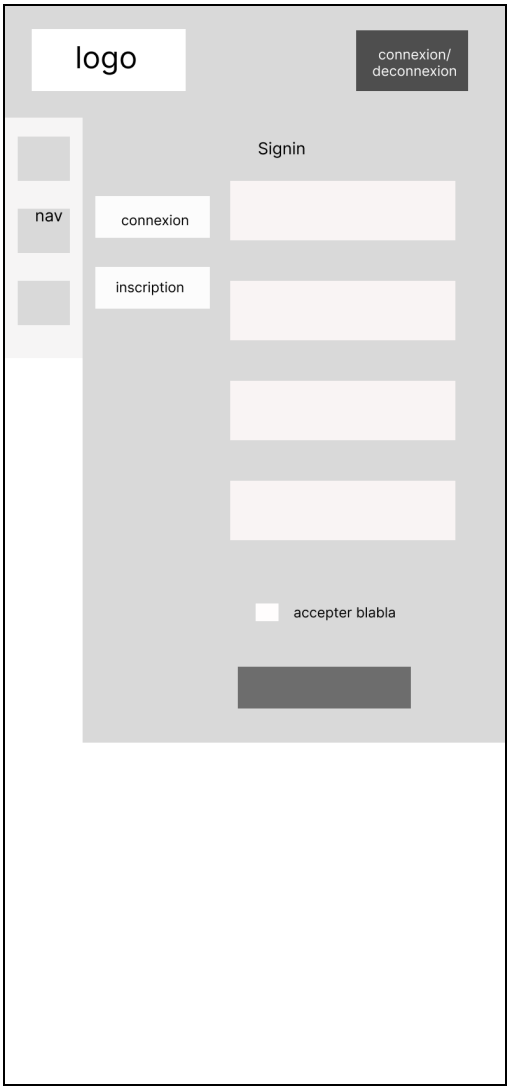
V. Wireframes

page Trends mobile

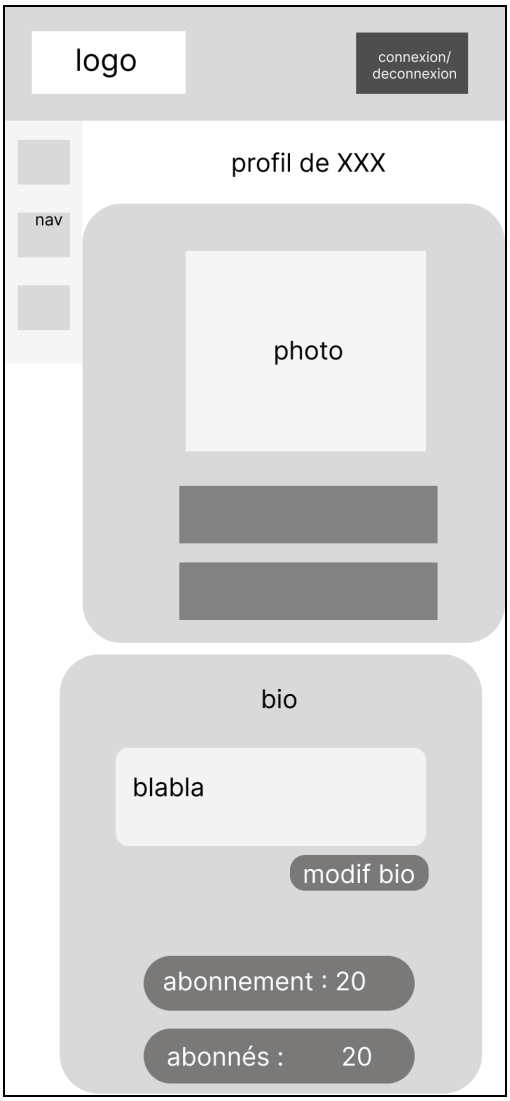


page d’Accueil Mobile

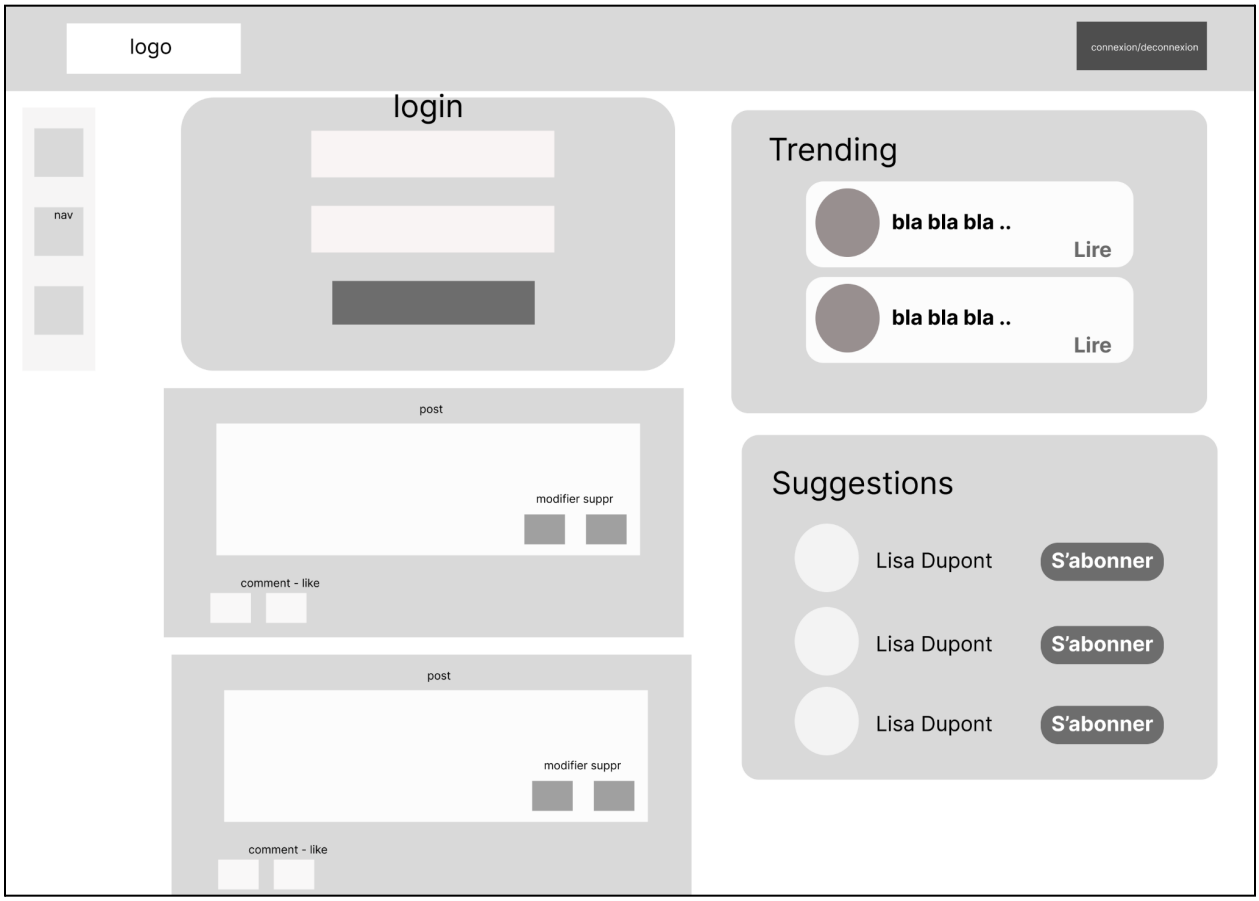




page Inscription/connexion

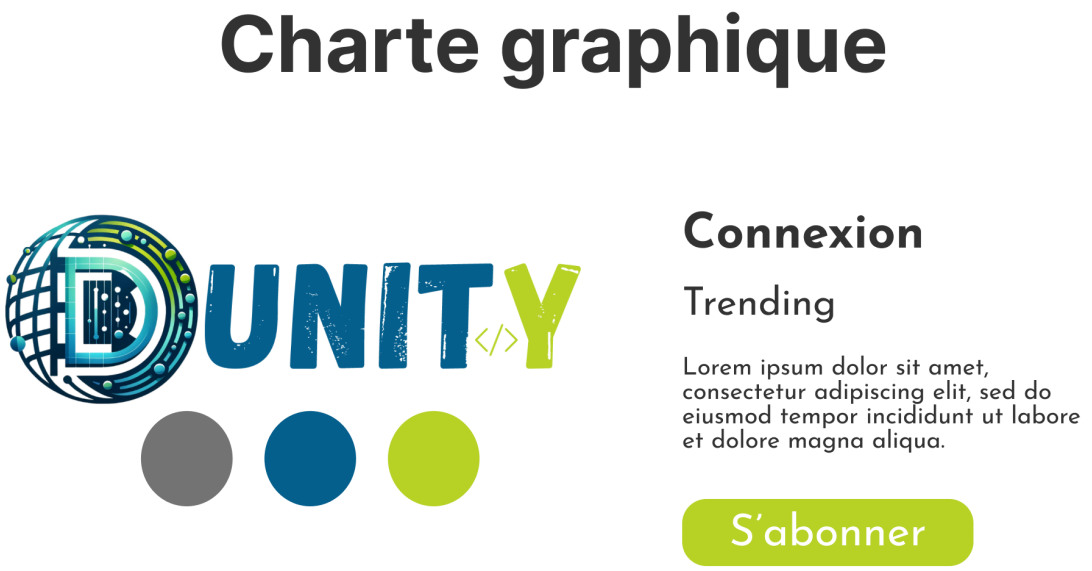


page Profil



Page d'accueil desktop

VI. Charte graphique et logo



Voici la charte graphique que j'ai concoctée pour donner une identité visuelle unique et marquante. Premièrement, le logo est un mélange ingénieux entre modernité et fonctionnalité. Il intègre le mot "UNITY" avec une diversité de couleurs et de formes, évoquant un monde digital, connecté et dynamique.

Parlons couleur maintenant. J'ai sélectionné trois couleurs principales. Un gris sobre, un bleu profond et un vert vibrant. Le gris pour son élégance et sa neutralité, le bleu évoquant la confiance et le professionnalisme, et le vert, symbole de créativité et d'innovation.

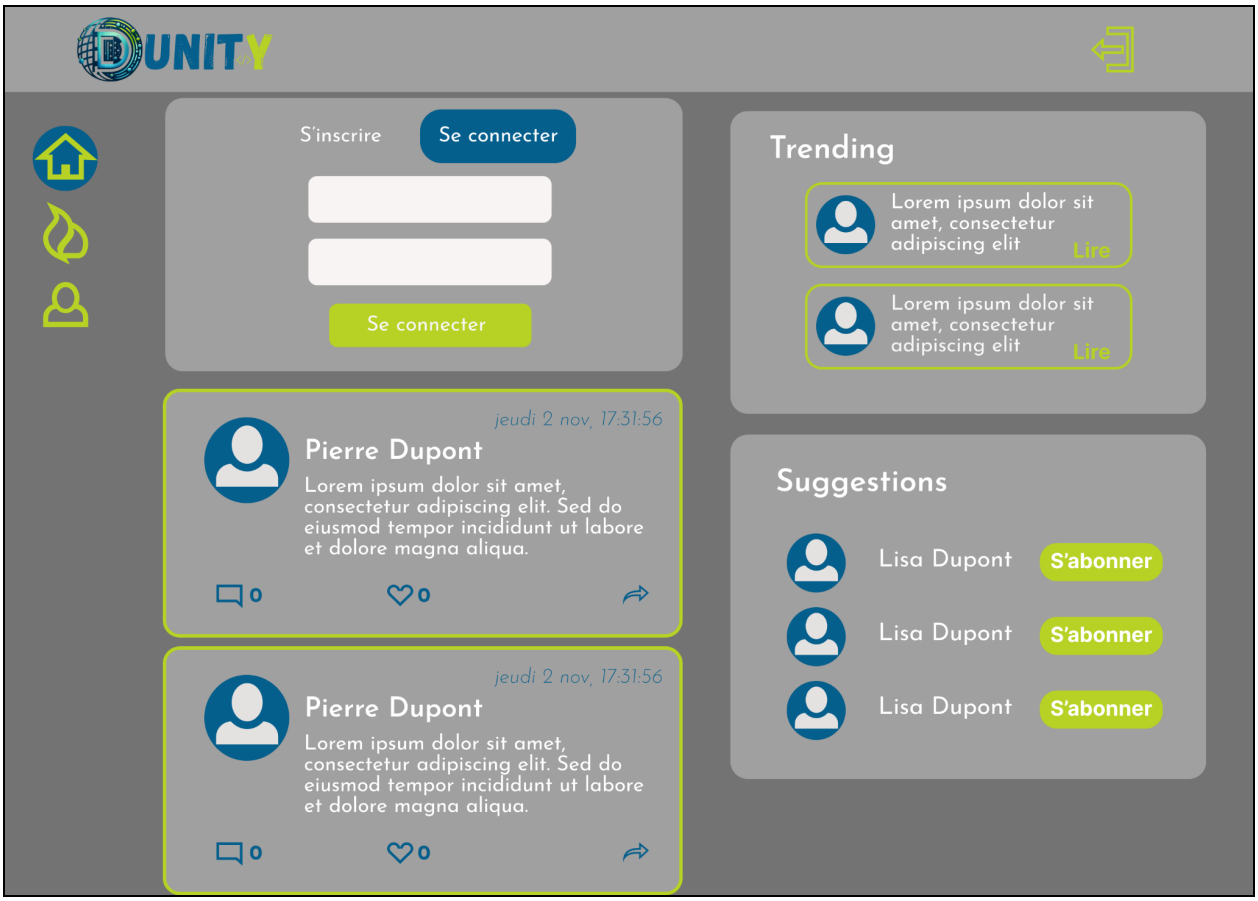
Le choix typographique n'est pas laissé au hasard. Une police moderne et épurée a été choisie pour garantir une lisibilité optimale et maintenir l'attention de l'utilisateur.

Enfin, les éléments de la page sont conçus pour être intuitifs et accessibles, garantissant une expérience utilisateur fluide et agréable. Le bouton "S'abonner", par exemple, est mis en évidence par une couleur vert vive, encourageant l'utilisateur à interagir.

En somme, cette charte graphique est une fusion réfléchie entre esthétique contemporaine et fonctionnalité intuitive, conçue pour engager efficacement l'utilisateur dans un environnement visuel cohérent. Et rappelez-vous, chaque détail a été pensé pour faire de votre interface une expérience visuelle mémorable et user-friendly.

VII. Exemples de maquettes

page d’accueil - Desktop



page d'accueil -Mobile

RÉALISATION PERSONNELLES

I. Upload d'image

Dans le cadre de ce projet, j'ai mis en place un système de téléchargement d'images en utilisant **Multer**, une bibliothèque middleware pour gérer les fichiers entrants dans les applications Node.js.

A. Back

```
if (!ObjectID.isValid(req.body.userId))
  return res.status(400).send("ID unknown : " + req.body.userId);
```

Le processus commence par la validation de l'identifiant de l'utilisateur avec ***ObjectID.isValid(req.body.userId)***. Si l'ID n'est pas valide, une réponse d'erreur est envoyée.

```
try {
  if (
    req.file.mimetype != "image/jpg" &&
    req.file.mimetype != "image/png" &&
    req.file.mimetype != "image/jpeg"
  ) {
    throw Error("invalid file");
  }

  if (req.file.size > 500000) throw Error("max size");
} catch (err) {
  const errors = uploadErrors(err);
  return res.status(200).json({ errors });
}
```

Ensuite, j'ai intégré une vérification du type de fichier et de sa taille. Si le fichier n'est pas une image JPG, PNG ou JPEG, ou si sa taille dépasse 500 Ko, une erreur est générée. Ces validations permettent de s'assurer que seuls des fichiers conformes et de taille raisonnable sont traités.

```
const fileName = req.body.name + ".jpg";

const uploadPath = __dirname + "/../client/public/uploads/profil/";
if (!fs.existsSync(uploadPath)) {
  fs.mkdirSync(uploadPath, { recursive: true });
}
```

Une fois ces vérifications passées, le nom de fichier est défini et le chemin d'upload est préparé. J'ai utilisé le module **fs** de **Node.js** pour vérifier l'existence du dossier de destination et pour le créer si nécessaire.

```
const readableStream = new Readable({
  read() {
    this.push(req.file.buffer);
    this.push(null);
  },
});

await pipeline(readableStream, fs.createWriteStream(uploadPath + fileName))
  .then(() => {
    console.log("image uploaded");
  })
  .catch((err) => {
    console.log(err);
  });
```

Le téléchargement de l'image est géré par la création d'un Readable Stream à partir du buffer de l'image, qui est ensuite écrit dans le système de fichiers du serveur via **pipeline**, une fonction promisifiée pour gérer les streams de façon efficace. Après le téléchargement réussi de l'image, son chemin est enregistré dans la base de données en associant l'image au profil utilisateur correspondant avec ***'userModel.findByIdAndUpdate'***.

```
try {
  await userModel
    .findByIdAndUpdate(
      req.body.userId,
      { $set: { picture: "./uploads/profil/" + fileName } },
      { new: true, upsert: true, setDefaultsOnInsert: true }
    )
    .then(() => {
      res.status(200).json({ message: "Successfully updated. " });
    })
```

Cette approche garantit non seulement la sécurité du processus d'upload, mais assure aussi que les données de l'utilisateur sont correctement mises à jour avec le lien vers la nouvelle image de profil.

B. Front

Pour cela, j'ai créé un composant React nommé **UploadImg**. Ce composant est conçu pour permettre aux utilisateurs de télécharger une nouvelle image de profil.

À l'intérieur de ce composant, j'utilise le hook ***useState*** pour gérer l'état local, spécifiquement pour stocker le fichier image sélectionné par l'utilisateur. J'ai aussi intégré **Redux** dans mon projet, en utilisant les hooks ***useDispatch*** pour envoyer des actions **Redux**, et ***useSelector*** pour accéder aux données utilisateur stockées dans le store Redux.

```
const [file, setFile] = useState();
const dispatch = useDispatch();
const userData = useSelector((state) => state.userReducer);
```

La logique principale pour le téléchargement de l'image se trouve dans la fonction ***handlePicture***, déclenchée lors de la soumission du formulaire. Cette fonction commence par prévenir le comportement par défaut de soumission, puis crée un objet **FormData**. Cet objet contient le fichier image, ainsi que le nom et l'ID de l'utilisateur, qui sont nécessaires pour l'API côté serveur.

```
const handlePicture = (e) => {
  e.preventDefault();
  const data = new FormData();
  data.append("name", userData.pseudo);
  data.append("userId", userData._id);
  data.append("file", file);

  dispatch(uploadPicture(data, userData._id));
  window.location.reload();
};
```

Après avoir préparé les données, j'appelle l'action **Redux *uploadPicture***, qui est responsable de l'envoi des données au serveur via une requête HTTP POST réalisée avec **Axios**. Cette requête inclut les cookies (avec ***withCredentials: true***) pour gérer correctement la session et l'authentification de l'utilisateur. Si l'envoi réussit, un deuxième appel **Axios** est effectué pour récupérer les informations mises à jour de l'utilisateur, y compris le nouveau lien de l'image de profil, qui sont ensuite intégrées dans le store **Redux**.

```
export const uploadPicture = (data, id) => {
  return (dispatch) => {
    return axios({
      withCredentials: true,
      method: "post",
      url: `${process.env.REACT_APP_API_URL}api/user/upload`,
      data,
    })
      .then((res) => {
        if (res.data.errors) {
          dispatch({ type: GET_USER_ERRORS, payload: res.data.errors });
        } else {
          dispatch({ type: GET_USER_ERRORS, payload: "" });
          return axios
            .get(`${process.env.REACT_APP_API_URL}api/user/${id}`)
            .then((res) => {
              dispatch({ type: UPLOAD_PICTURE, payload: res.data.picture });
            });
        }
      })
      .catch((err) => console.log(err));
```

En résumé, ce composant **React**, **UploadImg**, joue un rôle essentiel dans mon projet. Il offre une interface simple pour les utilisateurs souhaitant changer leur image de profil et utilise des techniques avancées telles que les requêtes asynchrones et la gestion d'état pour interagir avec le serveur. Cela m'a permis de comprendre comment les technologies front-end et back-end peuvent travailler ensemble pour améliorer l'expérience utilisateur.

II. Formulaire d'inscription

A. Back

J'ai développé un processus d'inscription des utilisateurs en utilisant un contrôleur dans mon backend **Node.js**. Ce contrôleur joue un rôle central dans la gestion des nouvelles inscriptions.

L'exécution commence avec la fonction ***signUp*** au sein de ce contrôleur. Cette fonction extrait d'abord les informations essentielles telles que l'email, le mot de passe et le pseudo à partir de la requête envoyée par l'utilisateur. C'est une étape cruciale pour récupérer les données nécessaires à la création d'un nouveau compte utilisateur.

```
signUp: async (req, res) => {
  const { email, password, pseudo } = req.body;
  const errors = await checkUserBeforeInsert(pseudo, email);
```

Une fois ces informations récupérées, une vérification importante est réalisée pour s'assurer de leur unicité et de leur conformité. Cela inclut des contrôles pour vérifier si l'email fourni n'est pas déjà utilisé par un autre compte. Si une anomalie est détectée à cette étape, comme un email déjà enregistré, une réponse d'erreur est immédiatement renvoyée au client pour informer de la situation.

```
const errors = await checkUserBeforeInsert(pseudo, email);
if (Object.values(errors).some((item) => item !== "")) {
  console.log(errors);
  return res.status(400).json({ errors });
}
```

En l'absence d'erreurs lors de cette vérification initiale, le processus avance avec la tentative de création de l'utilisateur via le modèle **Mongoose *userModel***. C'est à ce moment que le middleware défini dans le schéma de l'utilisateur intervient. Avant que

les informations de l'utilisateur ne soient enregistrées dans la base de données, ce middleware utilise la bibliothèque **bcrypt** pour hasher le mot de passe. Ce traitement assure une sécurité renforcée des données utilisateur, car le mot de passe stocké dans la base de données n'est jamais la version en clair mais sa forme hashée.

```
try {
  const user = await userModel.create({ email, password, pseudo });

userSchema.pre("save", async function (next) {
  const salt = await bcrypt.genSalt();
  this.password = await bcrypt.hash(this.password, salt);
  next();
});
```

Après le hashage du mot de passe, le processus de sauvegarde se poursuit normalement. Si l'enregistrement de l'utilisateur se déroule sans encombre, une réponse contenant l'ID du nouvel utilisateur est envoyée. Cette étape marque la réussite de l'inscription.

```
  res.status(201).json({ user: user._id });
} catch (err) {
  console.log(err);
  const errors = signUpErrors(err);
  res.status(400).json({ errors });
}
```

En revanche, si une erreur survient pendant la création de l'utilisateur, comme un problème de validation des données, les détails de cette erreur sont capturés. Un traitement spécifique est appliqué pour formater ces erreurs, et elles sont ensuite renvoyées au client. Cette gestion robuste des erreurs assure que tout problème est communiqué clairement et rapidement à l'utilisateur, contribuant ainsi à une expérience utilisateur fiable et transparente durant le processus d'inscription.

B. Front

Dans la partie frontale de mon projet, j'ai conçu le formulaire d'inscription des utilisateurs en utilisant **React**. J'ai créé un composant **SignUpForm** qui encapsule toute la logique nécessaire pour inscrire un nouvel utilisateur.

Au sein de ce composant, j'utilise des hooks d'état de **React** (***useState***) pour gérer les données saisies par l'utilisateur, comme le pseudo, l'email, le mot de passe, et la confirmation du mot de passe. J'ai également un état pour suivre si le formulaire a été soumis avec succès.

```
const [pseudo, setPseudo] = useState("");
const [email, setEmail] = useState("");
const [password, setPassword] = useState("");
const [controlPassword, setControlPassword] = useState("");
const [formSubmit, setFormSubmit] = useState(false);
```

Pour la validation des données du formulaire, j'ai mis en place un schéma de validation avec **Yup**. Ce schéma définit les règles pour chaque champ du formulaire, comme s'assurer que le pseudo n'est ni trop court ni trop long, que l'email est valide, et que les mots de passe sont corrects et correspondent. J'ai également inclus une validation pour la case à cocher des termes et conditions.

```
const validationSchema = Yup.object().shape({
  pseudo: Yup.string()
    .required("Le pseudo est requis")
    .min(3, "Trop court")
    .max(55, "Trop long"),
  email: Yup.string().email("Email invalide").required("L'email est requis"),
  password: Yup.string()
    .required("Le mot de passe est requis")
    .min(6, "Trop court")
    .max(1024, "Trop long"),
  controlPassword: Yup.string()
    .required("La confirmation est requise")
    .oneOf(
      [Yup.ref("password"), null],
      "Les mots de passe doivent correspondre"
    ),
  terms: Yup.boolean().oneOf(
    [true],
    "Veuillez accepter les conditions générales"
  ),
});
```

La logique de soumission du formulaire est gérée par la fonction ***handleRegister***. Lorsque l'utilisateur soumet le formulaire, cette fonction empêche d'abord le comportement de soumission par défaut, puis tente de valider les données avec le schéma **Yup**. Si les données sont valides, elles sont envoyées au serveur via une requête POST réalisée avec **Axios**. En cas de succès, l'état ***formSubmit*** est mis à jour pour indiquer que l'inscription a réussi, et l'utilisateur est invité à se connecter.

```
const handleRegister = async (e) => {
  e.preventDefault();

  // Sélectionner tous les éléments d'erreur et les réinitialiser
  const errorElements = document.querySelectorAll(".error");
  errorElements.forEach((element) => {
    element.innerHTML = "";
  });

  try {
    const terms = document.getElementById("terms");
    // Valider les données
    await validationSchema.validate(
      { pseudo, email, password, controlPassword, terms: terms.checked },
      { abortEarly: false }
    );

    // Si la validation est réussie, envoie les données
    await axios({
      method: "post",
      url: `${process.env.REACT_APP_API_URL}api/user/register`,
      withCredentials: true,
      data: { pseudo, email, password },
    });
    setFormSubmit(true);
```

En cas d'erreurs de validation, ces erreurs sont capturées et affichées à côté de chaque champ concerné. J'utilise des sélecteurs DOM pour trouver les éléments d'erreur correspondants et y insérer les messages d'erreur appropriés. Cette méthode assure une rétroaction instantanée et pertinente pour l'utilisateur, lui permettant de corriger ses saisies de manière efficace.

```
} catch (err) {
  const errors = {};

  // Parcourir chaque erreur et stocker uniquement la première pour chaque champ
  err.inner.forEach((error) => {
    // Si une erreur pour ce champ n'a pas déjà été enregistrée, l'enregistrer
    if (!errors[error.path]) {
      errors[error.path] = error.message;
    }
  });

  // Mettre à jour l'UI avec les erreurs
  for (const [path, message] of Object.entries(errors)) {
    const errorField = document.querySelector(`.${path}.error`);
    if (errorField) {
      errorField.innerHTML = message;
    }
  }
}
```

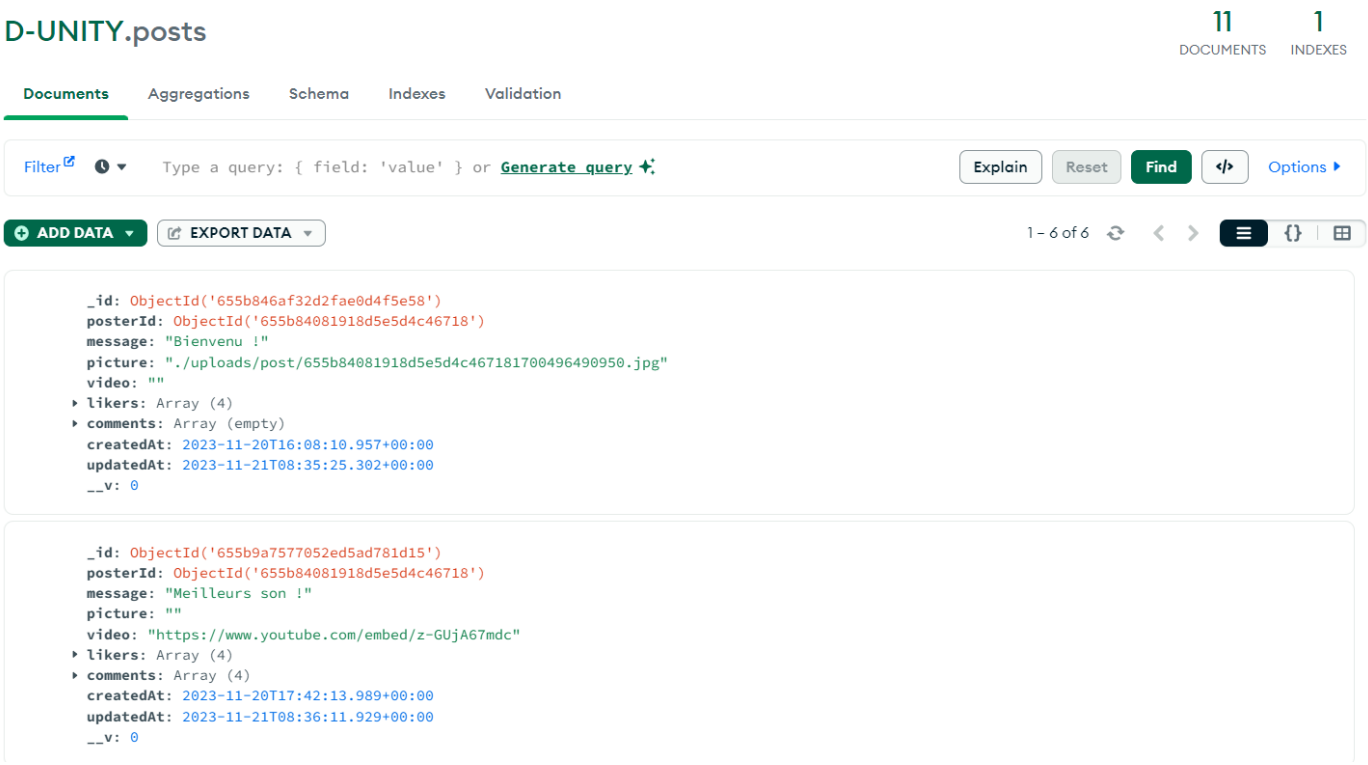
En résumé, le composant **SignUpForm** offre une interface utilisateur claire et intuitive pour l'inscription, avec des validations robustes et des retours d'erreur précis, garantissant une expérience utilisateur fluide et sécurisée lors de l'inscription sur la plateforme.

PRÉSENTATION DU JEU D'ESSAI

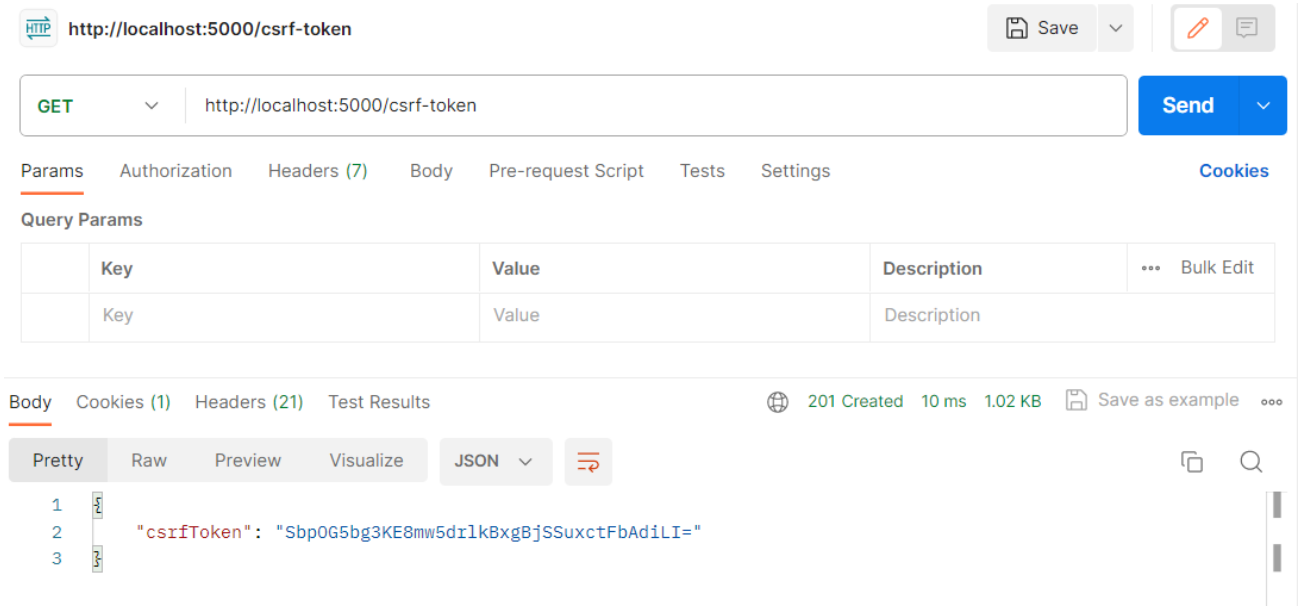
1. Test Fonctionnel

Pour illustrer la création d'un post avec une image téléversée depuis l'ordinateur du client, voici un jeu d'essai qui permet de vérifier le bon fonctionnement de la requête. Pour réaliser ce jeu d'essai, je vais me servir du logiciel Postman qui permet de tester, et de documents les requêtes vers une API.

Dans un premier temps, j'observe l'état actuel de notre base de données, avant l'ajout du post. La collection qui sera mise à jour lors de la création d'un post est la collection "Posts":

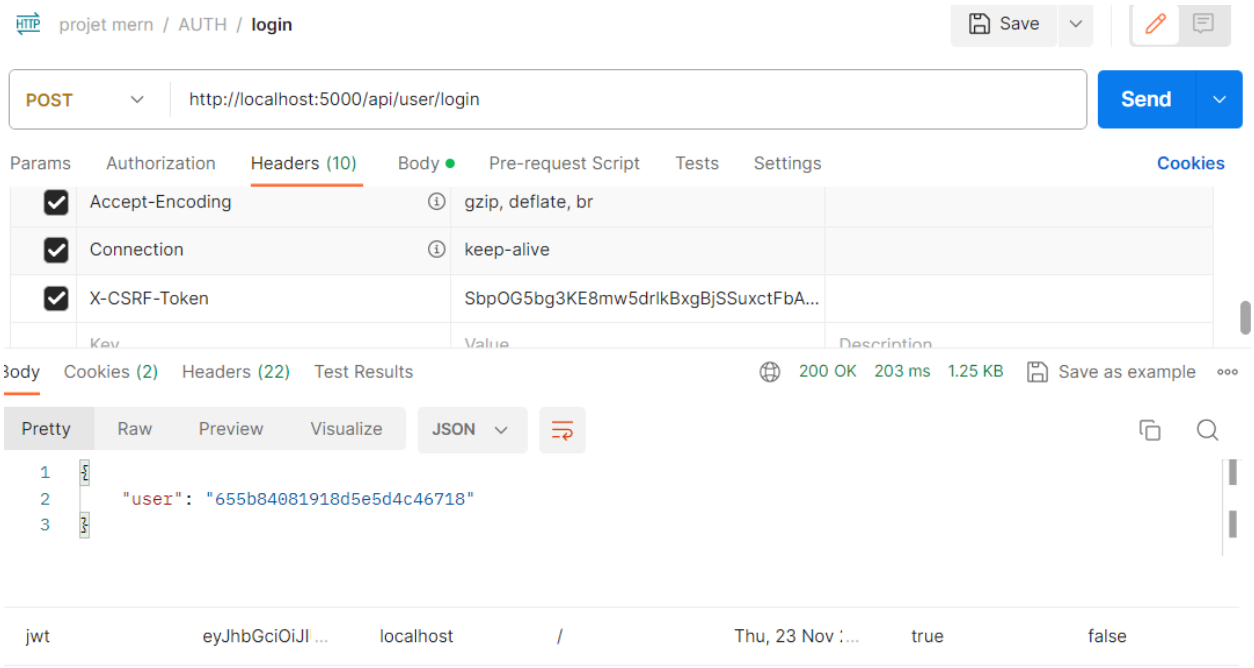


Les requêtes étant autorisée que pour les utilisateurs avec un cookie contenant le bon csrf, je dois tout d'abord récupérer un token csrf que je vais ensuite passer à ma requête de création d'un post. J'effectue donc une requête vers l'endpoint de récupération du token:



Ensuite, je viens créer la requête de connection, je vais tout d'abord ajouter le token récupéré dans l'onglet "Headers" de Postman

La requête de création d'un post n'étant autorisée que pour les utilisateurs connectés, je dois tout d'abord récupérer un JWT que je vais ensuite passer à ma requête de création d'un post. J'effectue donc une requête vers l'endpoint de récupération du token, avec mes identifiants.



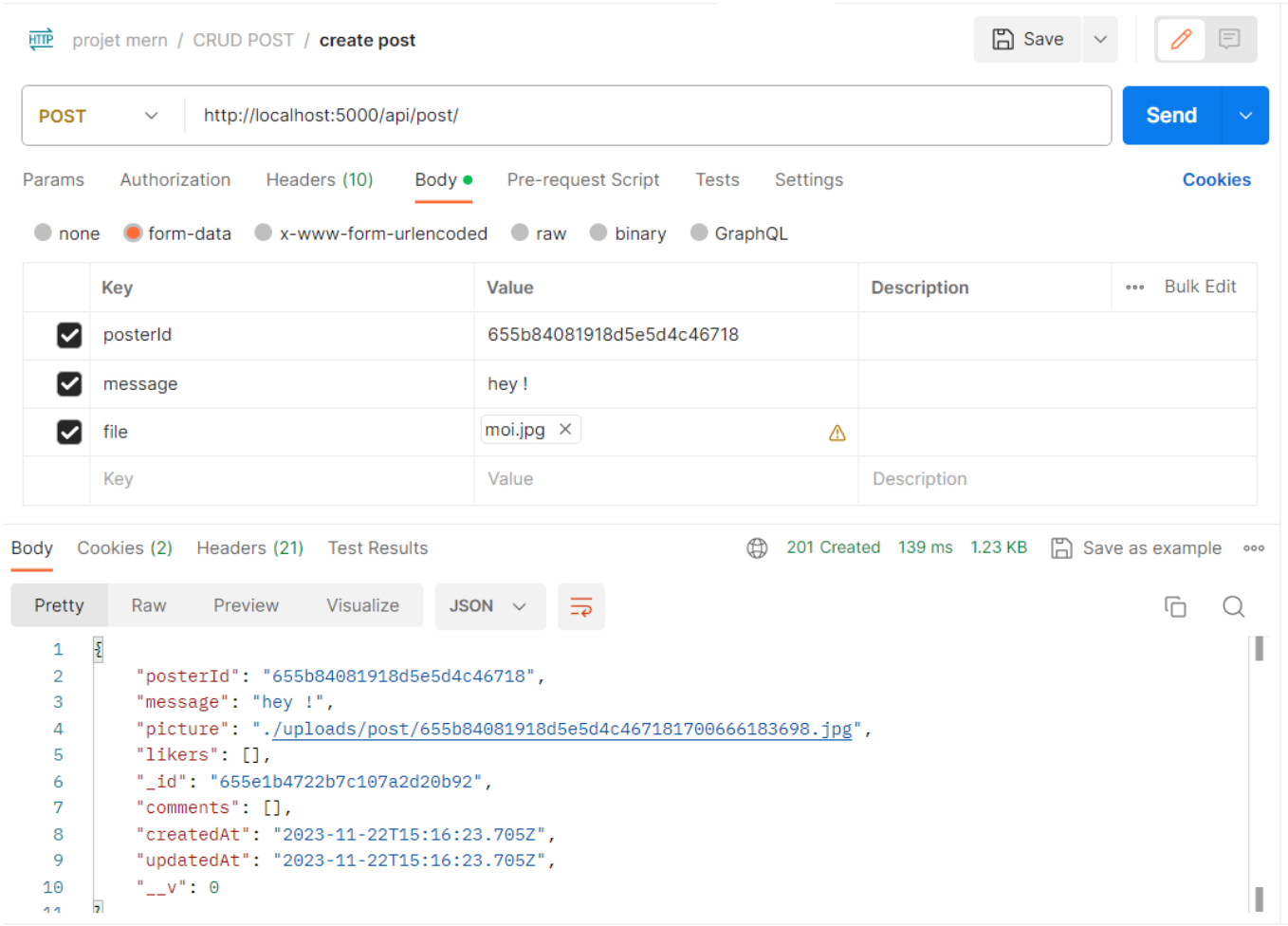
Ensuite, je viens créer la requête de création du post, avec le JWT récupéré dans les cookie de Postman.

Les données attendues lors de la création d'un post via l'API doivent respecter les critères suivants:

- Réponse au format JSON
- Un statut HTTP 201
- Les données entrées
- L'identifiant du post en base de données
- L'URL de l'image.

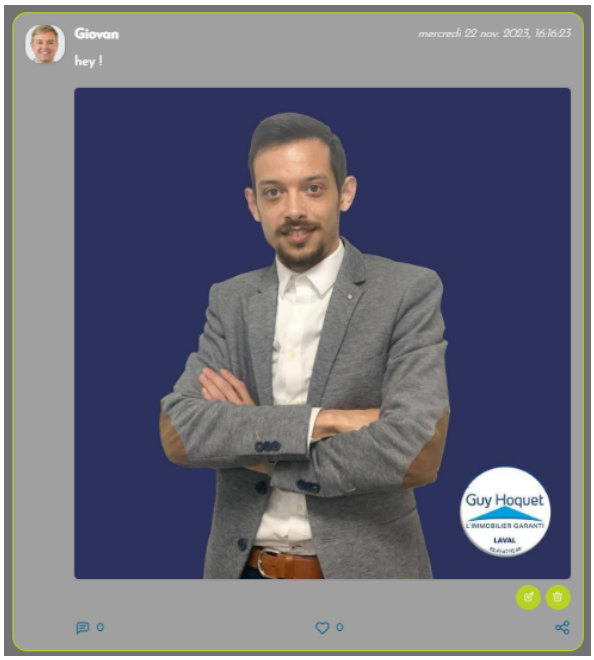
Ensuite, je passe à l'ajout des données que je souhaite envoyer à mon API. Pour ajouter une image je dois utiliser le type "formulaire", et non pas le type JSON comme pour le reste de mon API. Dans Postman, je vais alors sélectionner le type "form-data", dans l'onglet "Body". Ensuite, j'y ajoute les champs que je souhaite passer en paramètre, ainsi qu'un champ de type "file", avec le nom "image".

J'effectue ma requête et j'observe la réponse de mon API avec Postman:



La réponse de mon API est un statut HTTP 201 - Created, au format JSON. Le format JSON contient bien les données entrées, ainsi que l'ID du post. Pour vérifier que tout est correcte, je vais vérifier l'état de notre base de données, ainsi que sur l'application.

```
_id: ObjectId('655e1b4722b7c107a2d20b92')
posterId: ObjectId('655b84081918d5e5d4c46718')
message: "hey !"
picture: "./uploads/post/655b84081918d5e5d4c467181700666183698.jpg"
▸ likers: Array (empty)
▸ comments: Array (empty)
createdAt: 2023-11-22T15:16:23.705+00:00
updatedAt: 2023-11-22T15:16:23.705+00:00
__v: 0
```



2. Tests unitaires

Dans le cadre de mon projet, j'ai utilisé Jest, un framework de test JavaScript, pour écrire des tests unitaires pour différentes fonctions de gestion des erreurs et de validation dans mon application. Voici comment j'ai procédé :

Mise en Place des Tests :

J'ai commencé par définir les imports nécessaires, y compris les différentes fonctions de validation que je voulais tester (uploadErrors, signInErrors, signUpErrors, checkUserBeforeInsert) et le modèle utilisateur (userModel). J'ai ensuite simulé (mock) le modèle utilisateur pour isoler les tests des interactions réelles avec la base de données.

```
const { uploadErrors } = require("./errors.utils");
const { signInErrors } = require("./errors.utils");
const { signUpErrors } = require("./errors.utils");
const { checkUserBeforeInsert } = require("./errors.utils");
const userModel = require("../models/user.model");

jest.mock("../models/user.model");
```

- **Tests pour checkUserBeforeInsert :**

  J'ai écrit plusieurs cas de test pour cette fonction. D'abord, j'ai testé le cas

où le pseudo et l'email sont vides, m'attendant à recevoir des messages d'erreur spécifiques. Ensuite, j'ai testé le cas où le pseudo et l'email existent déjà dans la base de données. Pour cela, j'ai simulé la réponse du modèle utilisateur pour retourner true, signifiant que l'utilisateur existe. Enfin, j'ai testé le cas où le pseudo et l'email sont uniques et non vides, m'attendant à ne recevoir aucune erreur.

```
describe("checkUserBeforeInsert", () => {
  it("should return errors when pseudo and email are empty", async () => {
    const errors = await checkUserBeforeInsert(" ", " ");
    expect(errors).toEqual({
      pseudo: "Le pseudo est obligatoire",
      email: "L'email est obligatoire",
    });
  });
```

- **Tests pour signUpErrors :**

  Ces tests vérifient les erreurs de validation lors de l'inscription. J'ai testé différents scénarios en fonction du message d'erreur - par exemple, si le message contient "pseudo", "email", ou "password", et j'ai vérifié que les erreurs appropriées sont retournées. J'ai également testé un cas où aucun de ces termes n'est inclus dans le message d'erreur.

```
describe("signUpErrors", () => {
  it('should return pseudo error when error message includes "pseudo"', () => {
    const errors = signUpErrors({ message: "pseudo" });
    expect(errors).toEqual({
      pseudo: "Pseudo incorrect",
      email: "",
      password: "",
    });
  });
```

- **Tests pour signInErrors :**

  Ici, j'ai concentré mes tests sur les erreurs de connexion. J'ai vérifié que si le message d'erreur contient "incorrect", une erreur de login est retournée, et dans les autres cas, aucune erreur n'est retournée.

```
describe("signInErrors", () => {
  it('should return login error when error message includes "incorrect"', () => {
    const errors = signInErrors({ message: "incorrect" });
    expect(errors).toEqual({
      login: "Login incorrect",
    });
  });
```

- **Tests pour uploadErrors :**

  Ces tests couvrent les scénarios d'erreurs lors du téléchargement de fichiers. J'ai testé les cas où le message d'erreur indique un fichier invalide ou un fichier dépassant la taille maximale autorisée, ainsi qu'un cas où le message d'erreur ne contient ni l'un ni l'autre de ces termes.

```
describe("uploadErrors", () => {
  it('should return format error when error message includes "invalid file"', () => {
    const errors = uploadErrors({ message: "invalid file" });
    expect(errors).toEqual({
      format: "Format incompatible",
      maxSize: "",
    });
  });
```

Chacun de ces tests a été écrit avec soin pour s'assurer que les fonctions de validation fonctionnent correctement dans divers scénarios et retournent les erreurs attendues. Cela m'a permis de renforcer la fiabilité et la robustesse de mon application, en garantissant une gestion efficace des erreurs dans différents aspects de l'application.

VULNÉRABILITÉ DE SÉCURITÉ ET VEILLE

I. Veille technologique

A. Yup

1. Description

Yup est une bibliothèque de construction de schémas pour la validation d'objets en JavaScript. Légère et intuitive, elle est souvent utilisée pour valider des formulaires dans des applications React et Node.js. Elle permet de définir des schémas de validation avec une syntaxe concise et expressive, offrant une vaste gamme de validations intégrées.

2. Caractéristique

- **Définition de Schémas de Validation** : Yup permet de définir des schémas de validation pour différents types de données (comme les chaînes de caractères, les nombres, les dates) avec des règles de validation (obligatoire, longueur minimale, format spécifique, etc.).
- **Personnalisation des Messages d'Erreur** : Yup offre la possibilité de personnaliser les messages d'erreur pour chaque règle de validation, ce qui est essentiel pour une bonne expérience utilisateur.
- **Intégration avec D'autres Bibliothèques** : Yup s'intègre facilement avec d'autres bibliothèques de gestion de formulaires comme Formik ou React Hook Form, rendant la validation des données utilisateur encore plus fluide.

3. Utilisation

- **Validation de Formulaires dans React** : Yup est largement utilisé dans les applications React pour valider les données des formulaires avant leur soumission. Cela aide à prévenir l'envoi de données incorrectes ou incomplètes au serveur.
- **Validation Côté Serveur avec Node.js :** Yup peut également être utilisé côté serveur pour valider les données reçues dans les requêtes HTTP,

assurant ainsi que les données traitées par le serveur respectent les critères définis.

4. Exemple d'utilisation

```
const schema = Yup.object().shape({
  nom: Yup.string().required('Le nom est requis'),
  age: Yup.number().min(18, 'Vous devez avoir au moins 18 ans')
});
```

5. Conclusion

L'utilisation de Yup dans les projets de développement web modernes souligne l'importance d'une validation robuste des données côté client et serveur. Sa simplicité d'utilisation, couplée à sa flexibilité, en fait un outil incontournable pour les développeurs soucieux de la qualité et de la sécurité des données dans leurs applications.

B. Axios

1. Description

Axios est une bibliothèque populaire en JavaScript utilisée pour effectuer des requêtes HTTP. Elle fonctionne à la fois dans un environnement de navigateur et avec Node.js, ce qui en fait un outil polyvalent pour les développeurs web. Basée sur les promesses, Axios facilite l'écriture de code asynchrone clair et compréhensible.

2. Caractéristiques

- **Support des Promesses** : Axios repose sur le concept des promesses, offrant une gestion simplifiée et élégante des opérations asynchrones.
- **Interception des Requêtes et des Réponses** : Permet de manipuler les requêtes et les réponses avant leur traitement final, ce qui est utile pour la configuration des en-têtes, la gestion des jetons d'authentification, etc.
- **Protection contre les attaques CSRF/XSRF** : Axios fournit des mécanismes automatiques pour protéger contre les attaques CSRF/XSRF, une considération importante pour la sécurité des applications web.

3. Utilisation

- **Communication avec les API REST** : Axios est largement utilisé pour interagir avec des API REST, envoyant des requêtes HTTP pour récupérer ou envoyer des données.
- **Applications React et Vue.js** : Dans les applications frontend modernes, Axios est souvent utilisé pour la communication entre le frontend et le backend, en particulier avec des frameworks comme React et Vue.js.
- **Développement de Serveurs Node.js** : Bien que principalement utilisé sur le front-end, Axios est également applicable dans des environnements Node.js, pour des tâches telles que la consommation d'API tierces.

4. Exemple d'utilisation

```
axios.get('https://api.exemple.com/data')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error(error);
  });
```

5. Conclusion

Axios est un outil essentiel pour les développeurs JavaScript modernes, offrant une méthode efficace et sécurisée pour les requêtes HTTP. Sa capacité à gérer des requêtes asynchrones avec une syntaxe épurée et des fonctionnalités robustes en fait un choix privilégié pour de nombreux projets web

II. Veille de sécurité

A. JWT et cryptage du mot de passe utilisateur

Pour réaliser le système de connexion de l'utilisateur, j'ai choisi d'utiliser les Json Web Tokens (JWT), je me posais également la question de la sécurité du stockage des mots de passe des utilisateurs en base de données.

Après avoir effectué quelques recherches sur les façon de stocker les mots de passe, j'ai décidé de les hasher en utilisant la librairie bcrypt.

- https://blog.logrocket.com/password-hashing-node-js-bcrypt/

-https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#hashing-vs-encryption

Puis j'ai effectué des recherches sur les bonnes pratiques à adopter en matière de sécurisation des JWT:

- https://curity.io/resources/learn/jwt-best-practices/

-https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html#jwt

-https://stackoverflow.com/questions/27301557/if-you-can-decode-jwt-how-are-they-secure

Enfin, j'ai cherché à savoir quelles sont les bonnes pratiques en matière de transmission du JWT entre le front et le back.

-https://levelup.gitconnected.com/bearer-token-authentication-and-authorization-6e4d1689083354/59

B. Sécurisation des requêtes

Lors de la mise en place de la sécurité pour le backend de mon application, la question de l'utilisation de CSRF (Cross-Site Request Forgery) avec le middleware Lusca a été primordiale. Pour bien comprendre son importance et son fonctionnement, j'ai mené des recherches approfondies.

Node.js CSRF protection middleware

OWASP CSRF Prevention Cheat Sheet

Understanding CSRF

Ces ressources m'ont aidé à saisir que Lusca offre une protection robuste contre les attaques CSRF en gérant les tokens de validation de manière efficace. Dans chaque requête soumise par le frontend, un token CSRF est intégré et vérifié par le serveur lors de la réception de la requête. Cette vérification empêche qu'un site tiers puisse effectuer des actions non autorisées au nom de l'utilisateur.

Lusca s'est révélé être un outil puissant pour renforcer la sécurité de mon application, en assurant que chaque requête est légitime et provient bien de l'interface utilisateur

autorisée, protégeant ainsi l'application contre les tentatives d'exploitation de vulnérabilités CSRF

J'aurais souhaité mettre en place un système de rate-limiting au niveau de notre API, ainsi que la mise en place de règles plus contraignantes au niveau des CORS. Au vu du temps qui m'était imparti pour réaliser ce projet, je n'ai pas eu le temps de procéder à ces ajouts, mais ces ajouts pourraient être intégrés dans une mise à jour de mon application qui respecterait au maximum les bonnes pratiques de sécurité établies par la fondation OWASP.

- https://www.cloudflare.com/learning/bots/what-is-rate-limiting/

- https://auth0.com/blog/cors-tutorial-a-guide-to-cross-origin-resource-sharing/

- https://cheatsheetseries.owasp.org/

III. Processus de recherche et traduction

A. Processus

Pour effectuer mes recherches, j'adopte en grande partie le processus de recherche suivant:

1. Requête en anglais, plutôt qu'en français, sur google avec des termes simples

2. Sélection des sources par rapport au classement personnel, ainsi que la date de création / mise à jour de la source

3. Analyse critique des informations contenues dans les pages sélectionnées

En règle général je sélectionne les ressources dans l'ordre suivant:

1. La documentation de la technologie

2. Un acteur ou un groupe d'acteurs connus et identifiés dans le domaine de la recherche, par exemple la fondation OWASP dans le cadre d'une recherche sur la sécurité

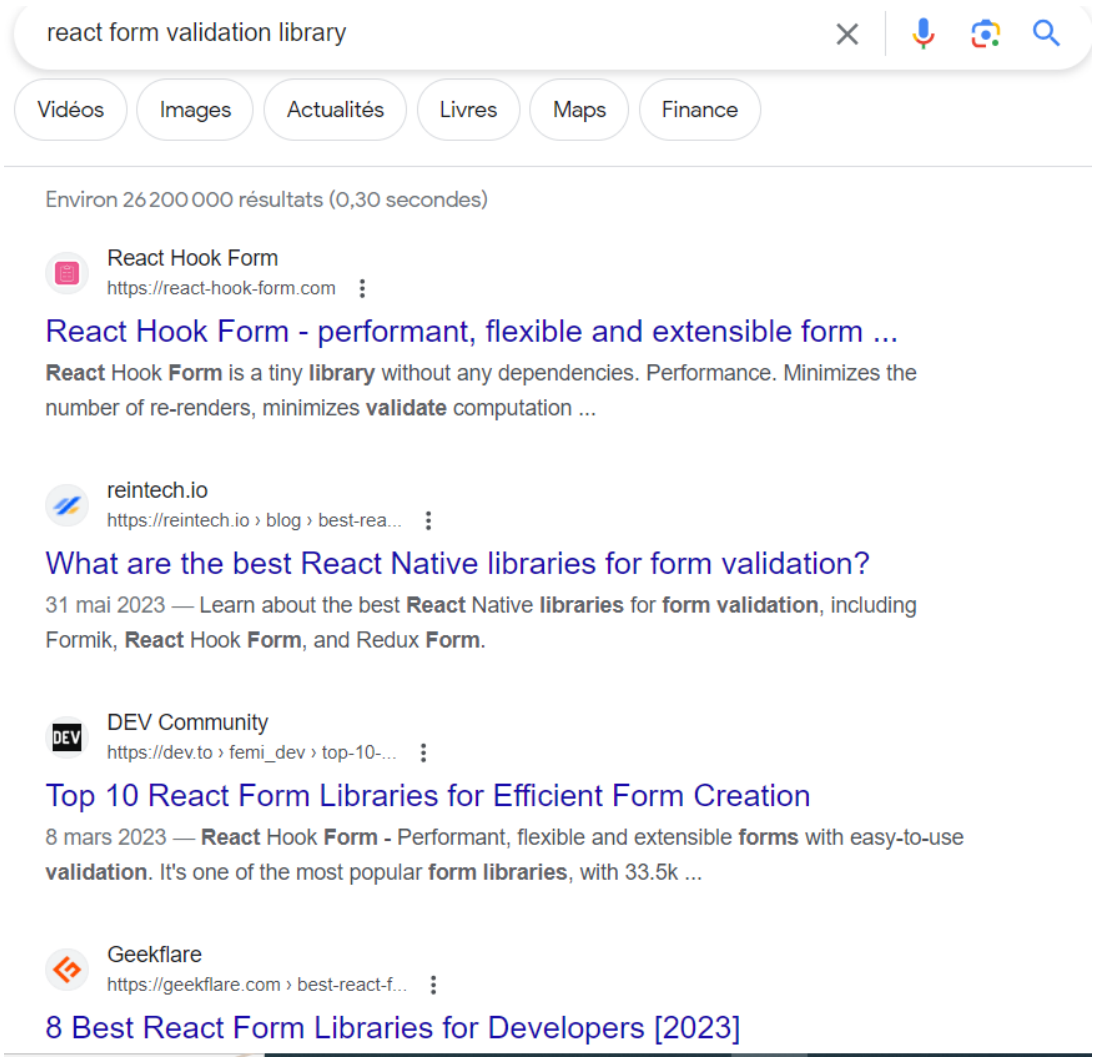
3. Les sites ayant attrait à la programmation, exemple: medium, blog LogRocket, etc.

4. Éventuellement des questions sur Stackoverflow, mais je préfère éviter Stack

> Overflow quand il s'agit de questions liées au code en lui-même, cependant j'apprécie les différences de points de vue sur des questions "théoriques".

Pour illustrer cette pratique, je vais reprendre l'interrogation que je me posais sur les librairies de validation de formulaire avec React.

Dans un premier temps, j'effectue une requête simple pour aborder ce questionnement:



Ensuite j'observe les résultats, la première est la documentation de React Hook Form, je garde cependant à l'esprit que ce sujet est ouvert à interprétation je décide donc de consulter cette page de la documentation de React Hook Form.

Mais ce sujet étant source de débats, je regarde également les questions de Stack Overflow pour tenter d'avoir d'autres points de vue.

B. Ressource en anglais

Dans le cadre de cette recherche, voici l'extrait que j'ai choisi de traduire. Il est issu de la documentation de React Hook From :

React Hook Form is a library for managing forms in React applications. It simplifies form handling by reducing the amount of code needed and improving performance. The library utilizes React hooks to manage form states and validations.

Key Features:

- **Efficient Form Handling**: React Hook Form minimizes re-renders and optimizes performance, especially for forms with a large number of fields.
- **Easy Integrations**: It works well with other libraries, including Yup for validation schemas.
- **Built-in Validation**: Supports both HTML5 native validation and custom validation rules.
- **Controlled Components**: It works seamlessly with controlled components, making it easy to integrate with UI libraries.

React Hook Form is ideal for developers who want a lightweight solution for form handling in React, offering a more straightforward and performance-optimized approach compared to traditional methods.

C. Traduction effectuée

React Hook Form est une bibliothèque destinée à la gestion des formulaires dans les applications React. Elle simplifie la manipulation des formulaires en réduisant la quantité de code nécessaire et en améliorant les performances. La bibliothèque utilise les hooks de React pour gérer les états et les validations des formulaires.

Caractéristiques Principales :

- **Gestion Efficace des Formulaires** : React Hook Form minimise les re-renders et optimise les performances, ce qui est particulièrement bénéfique pour les formulaires avec de nombreux champs.

- **Intégrations Faciles** : Elle s'intègre bien avec d'autres bibliothèques, y compris Yup pour les schémas de validation.
- **Validation Intégrée** : Supporte la validation native HTML5 ainsi que des règles de validation personnalisées.
- **Composants Contrôlés** : Elle fonctionne de manière transparente avec les composants contrôlés, facilitant ainsi l'intégration avec les bibliothèques UI.

React Hook Form est idéale pour les développeurs recherchant une solution légère pour la gestion des formulaires dans React, offrant une approche plus simple et optimisée en termes de performances par rapport aux méthodes traditionnelles.

CONCLUSION

En conclusion, ce projet a été extrêmement enrichissant pour moi. J'ai grandement apprécié le développement de cette application. L'émulation d'idées et de techniques, combinée à mon expérience antérieure, a facilité la mise en place rapide et efficace d'une méthodologie de travail qui m'a accompagné tout au long de ce mois de création. Ce processus a été crucial pour répondre à une problématique que j'avais personnellement rencontrée.

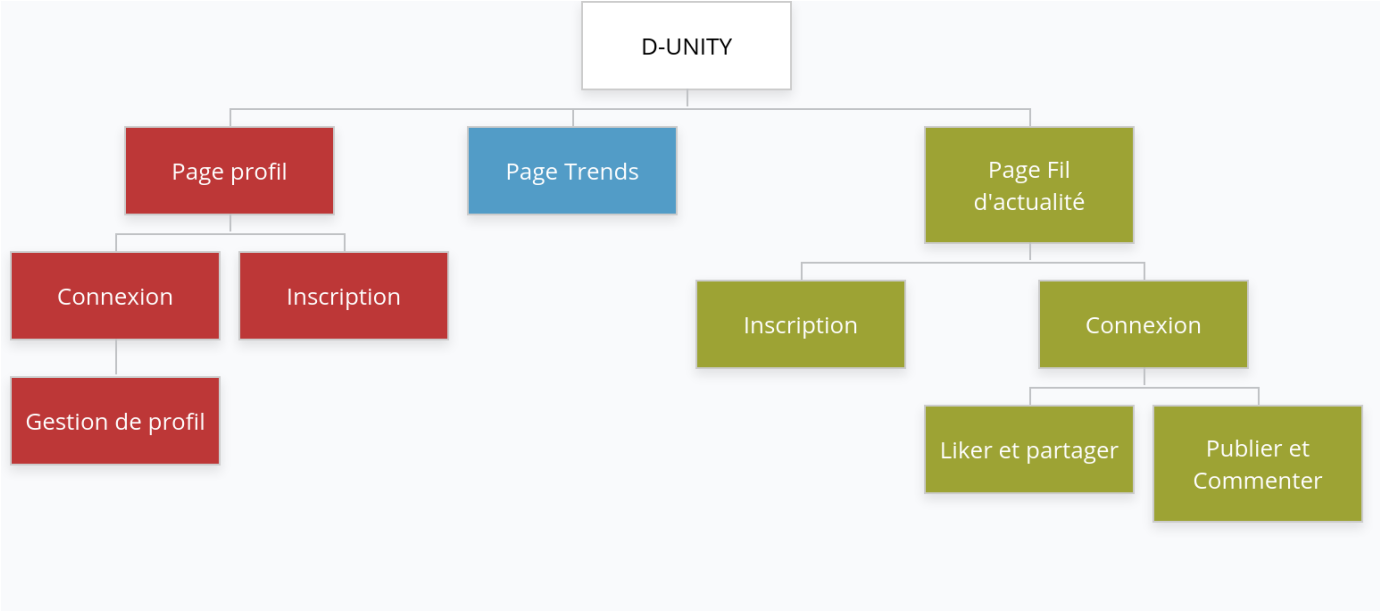
La mise en place rapide de solides bases de gestion m'a permis de réfléchir et d'élaborer des solutions adaptées à la problématique de la gestion des actions en temps réel sur l'interface utilisateur. Bien que le projet m'ait présenté des défis techniques, notamment en abordant des technologies que je n'avais peu ou pas utilisées auparavant, j'ai réussi à combiner la recherche sur ces technologies avec le développement de mon projet.

Initialement, j'appréhendais cette phase de construction en solitaire, étant donné la durée du projet, à la fois longue et brève au regard de l'ampleur de l'ambition. Toutefois, à l'issue de cette période, j'étais pleinement convaincu de l'importance de cette expérience et des bonnes pratiques qu'elle m'a inculquées, enrichissant ainsi mon parcours professionnel.

ANNEXES

Annexe 1

Arborescence



Annexe 2

Graphique users Strories