Groupomania

Projet n°7

Présentation des livrables

❖ Création du repo Github contenant:

- un dossier Back-End.
- un dossier Front-End.
- Les 3 fichiers de la base de donnée SQL.

❖ Afin de faire fonctionner l'API :

- Lancer dans le dossier FRONT et le dossier BACK => npm start
- Dans le dossier Back-End, exécuter via le terminal => nodemon server
- Dans le dossier Front-End, exécuter via le terminal => npm run serve

Accessibilité et SEO

http://localhost:8080/publication

Performances	Accessibilité	Bonnes pratiques	SEO	PWA
56	97	100	100	—

56

Performances

Les valeurs sont estimées et peuvent varier. Le calcul du score lié aux performances repose directement sur ces statistiques. Affichez la calculatrice.

▲ 0–49 ■ 50–89 ● 90–100

Création de la base de donnée

- Mise en place de table utilisateur, publication et commentaire ayant une relation 1 à plusieurs et la mise en place d'un delete en cascade.

```sql
CREATE TABLE `utilisateur` (
  `id` int NOT NULL AUTO_INCREMENT,
  `nom` varchar(100) DEFAULT NULL,
  `prenom` varchar(100) DEFAULT NULL,
  `pseudo` varchar(100) NOT NULL,
  `email` varchar(255) NOT NULL,
  `password` varchar(100) NOT NULL,
  `imageUrl` text,
  `role` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `pseudo` (`pseudo`),
  UNIQUE KEY `email` (`email`)
```

```sql
CREATE TABLE `publication` (
  `id` int NOT NULL AUTO_INCREMENT,
  `titre` varchar(150) NOT NULL,
  `message` text,
  `utilisateur_id` int NOT NULL,
  `imageUrl` text,
  PRIMARY KEY (`id`),
  KEY `utilisateur_id` (`utilisateur_id`),
  CONSTRAINT `publication_ibfk_1` FOREIGN KEY (`utilisateur_id`) REFERENCES `utilisateur` (`id`) ON DELETE CASCADE
```

```sql
CREATE TABLE `commentaire` (
  `id` int NOT NULL AUTO_INCREMENT,
  `message` text,
  `publication_id` int NOT NULL,
  `utilisateur_id` int NOT NULL,
  `imageUrl` text,
  PRIMARY KEY (`id`),
  KEY `publication_id` (`publication_id`),
  KEY `utilisateur_id` (`utilisateur_id`),
  CONSTRAINT `commentaire_ibfk_1` FOREIGN KEY (`publication_id`) REFERENCES `publication` (`id`) ON DELETE CASCADE,
  CONSTRAINT `commentaire_ibfk_2` FOREIGN KEY (`utilisateur_id`) REFERENCES `utilisateur` (`id`) ON DELETE CASCADE
```

Sécurité dans la base de donnée

- Les mots de passes sont cryptés avec Bcrypt et l'accès à la base de donnée et enregistré dans un fichier .env.sample en Back-End que j'ai rendu visible uniquement pour la soutenance.

id	nom	prenom	pseudo	email	password	imageUrl	role
1	Groupomania	Communication	Modérateur(trice)	Contact@Groupomania.fr	$2b$10$asec5r.uP4qp67U732/H9exU.TKQbxw...	http://localhost:3000/images/bulb-g87c9e6983_1920.jpg16...	ADMIN
2	test	test	test	test@test.fr	$2b$10$wIDIK13hBzswy/s.ahmcf.4ouRMRQpqI...	http://localhost:3000/images/animal-g0a1e5a922_1920.jpg...	UTILISATEUR
4	truc	truc	truc	truc@truc.fr	$2b$10$sgGT9bCtj0ebL9xlQyIA4.Igt3Bca4Qhg...	http://localhost:3000/images/northern-lights-g510b9836c_1...	UTILISATEUR
15	rossignol	Gaetan	Granstek	xynfsu@aol.com	$2b$10$CJLh5SRxw3c3EcxR6RAhM.xV7VCMKN...	http://localhost:3000/images/utilisateur_defaut.jpg	UTILISATEUR
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Mise en place du Back-End et de Sequelize 1/3

- J’utilise un Serveur **Node.js et le** Framework **Express.js,** ainsi que **Sequelize** pour communiquer avec la base de donnée SQL.
- Le fichier environnement contient les accès à la base de donnée qui sont utiliser dans le fichier de config de sequelize pour s’identifier puis la connexion s’effectue dans l’index.js situé dans le dossier models.

.env.sample

```
You, il y a 3 jours | 1 author (You)
bdd_nom = "groupomania"
bdd_login = "root"
bdd_password = "Groupomania!"
bdd_host = "localhost"
```

```
const { sequelize } = require("../config/sequelize.js");
// Connexion à la base de donnée MySQL
sequelize.authenticate()
    .then(() => {
        console.log("Connection établi avec succès.");
    })
    .catch((err) => {
        console.error("Impossible de se connecter à la BDD:", err);
    });
```

```
const Sequelize = require("sequelize");

// Pour lire les fichiers .env
require("dotenv").config({ path: process.cwd() + "/.env.sample" });

console.log("Get connection ...");

const sequelize = new Sequelize({
  database: process.env.bdd_nom,
  username: process.env.bdd_login,
  password: process.env.bdd_password,
  host: process.env.bdd_host,
  dialect: "mysql",
});

//on exporte pour utiliser notre connexion depuis les autres fichiers.
var exports = (module.exports = {});
exports.sequelize = sequelize;
```

Mise en place du Back-End et de Sequelize 2/3

- Pour que Sequelize puisse fonctionner correctement j'ai mis en place les liaisons des tables de la base de donnée.

- J'y explique que l'utilisateur peut avoir plusieurs publication et commentaire.

- Que les publications peuvent avoir plusieurs commentaires.

- Que les commentaires dépendent d'un utilisateur et d'une publication et que la publication dépend d'un utilisateur.

- Je spécifie la suppression en cascade.

```
// Modèles et tables
db.User = require("./user.js");
db.Publication = require("./publication.js");      You, la semaine dernière •
db.Commentaire = require("./commentaire.js");


// Relations entre les différentes tables
db.User.hasMany(db.Publication, { foreignKey: 'utilisateur_id' });
db.User.hasMany(db.Commentaire, { foreignKey: 'utilisateur_id' });

db.Publication.belongsTo(db.User, { foreignKey: 'utilisateur_id' }, {
    onDelete: 'CASCADE',
    hooks: true
});
db.Publication.hasMany(db.Commentaire, { foreignKey: 'publication_id' });

db.Commentaire.belongsTo(db.User, { foreignKey: 'utilisateur_id' });

db.Commentaire.belongsTo(db.Publication, { foreignKey: 'publication_id' }, {
    onDelete: 'CASCADE',
    hooks: true
});

module.exports = db
```

Mise en place du Back-End et de Sequelize 3/3

- Je met ensuite en place mes routes, mes models et mes controllers.

```
router.post("/signup", userCtrl.signup);
router.post("/login", userCtrl.login);

// supprimer un compte existant dans la base de donnée
router.delete("/delete/:id", auth, multer, userCtrl.deleteUtilisateur);

// Information de profil
router.get("/profil/:id", auth, multer, userCtrl.userProfil);
      You, la semaine dernière • Corrections massives du back-end, mise
// Information de tous les profils
router.get("/profil/", auth, multer, userCtrl.userProfilAll);

// mettre à jour un compte existant dans la base de donnée
router.put("/update/:id", auth, multer, userCtrl.updateUtilisateur);
```

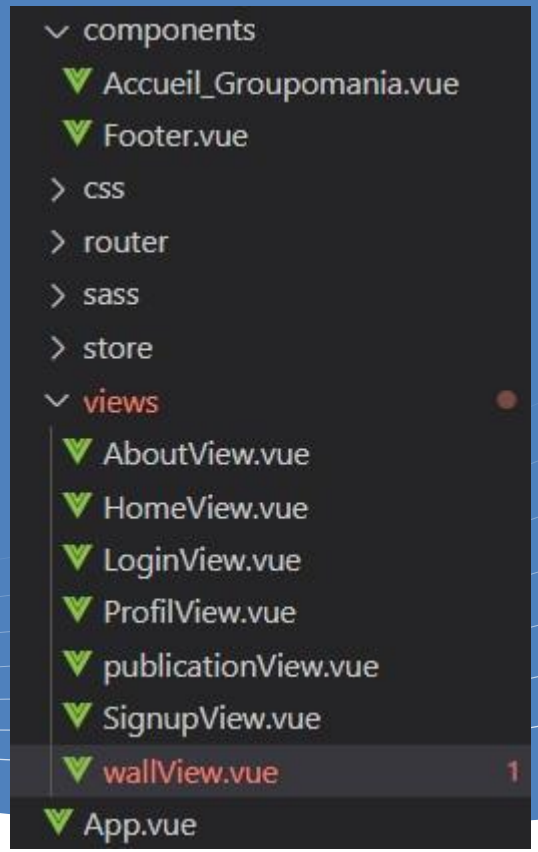
```
const { Sequelize, DataTypes, Model } = require("sequelize");
const { sequelize } = require("../config/sequelize.js");

// model pour l'utilisateur
const User = sequelize.define(
  "utilisateur",
  {
    id: { type: Sequelize.BIGINT, primaryKey: true, autoIncrement: true },
    nom: { type: Sequelize.STRING, allowNull: false },
    prenom: { type: Sequelize.STRING, allowNull: false },
    pseudo: { type: Sequelize.STRING, allowNull: false, unique: true },
    email: { type: Sequelize.STRING, allowNull: false, unique: true },
    password: { type: Sequelize.STRING, allowNull: false },
    imageUrl: {
      type: Sequelize.STRING,
    },
    role: {
      type: Sequelize.STRING,      You, le mois dernier • mise à jour mod
      allowNull: false,
      defaultValue: "UTILISATEUR",
    },
  },
  {
    freezeTableName: true,
    timestamps: false,
  }
);
```

Sécurité Back-End

- Xss clean => permet de nettoyer les entrées utilisateur provenant du corps POST, des requêtes GET et des paramètres d'URL, Protection contre les attaques XSS.
- Helmet => permet de sécurisé les en-têtes HTTP.
- Password-validator => permet d'imposer une complexité pour valider un mot de passe en définissant des règles.
- Bcrypt => permet que le mot de passe de l'utilisateur soit hashé.
- L'authentification est demandé sur toutes les routes.
- Jsonwebtoken => permet de créer et contrôler les tokens.
- Les versions les plus récentes des logiciels sont utilisées avec des correctifs de sécurité actualisés.

Mise en place du Front-End

- J’utilise **Vue.js V3.**
- **Axios** pour effectuer mes requêtes.
- Ainsi que **Sass** pour le CSS.
- Je crée ensuite mes vues correspondant à mes pages et je met en place les routes qui y mènerons.




```
const routes = [
  {
    path: '/',
    name: 'Accueil',
    component: HomeView
  },
  {
    path: '/about',
    name: 'A propos',
    component: AboutView
  },
  {      You, il y a 3 semaines •
    path: '/signup',
    name: 'Inscription',
    component: SignupView
  },
  {
    path: '/login',
    name: 'Connexion',
    component: LoginView
  },
  {
    path: '/profil',
    name: 'Profil',
    component: ProfilView
  },
  {
    path: '/publication',
    name: 'mur de l entreprise',
    component: wallView
  },
  {
    path: '/publication/:id',
    name: 'publication',
    component: publicationView
```

Axios

- Par exemple ici j'utilise axios afin de récupérer avec la méthode GET toutes les publications présentent dans ma base de donnée directement à l'arrivée sur ma page /publication.

- Les publications sont renvoyées par le Back-End directement en ordre descendant avec des informations sur les utilisateurs limitées.

```
const axios = require("axios");
const instancePost = axios.create({
  baseURL: "http://localhost:3000/api/",
});
```

```
exports.getAllPublication = (req, res, next) => {
  // findByPk pour trouver tous les objets
  Publication.findAll({
    order: [
      ['id', 'DESC'],
    ],
    include: [
      { model: User, required: true, attributes: ['nom', 'prenom', 'pseudo',
      'imageUrl'] },
      { model: Commentaire, required: false, include: [{ model: User, attributes:
      ['nom', 'prenom', 'pseudo', 'imageUrl'] }] }]
  })
    // récupération du tableau de tous les publications, et ont renvoi le tableau
    par le Back-End (base de donnée)
    .then((publications) => res.status(200).json(publications))
    .catch((error) => res.status(500).json({ error }));
};
```

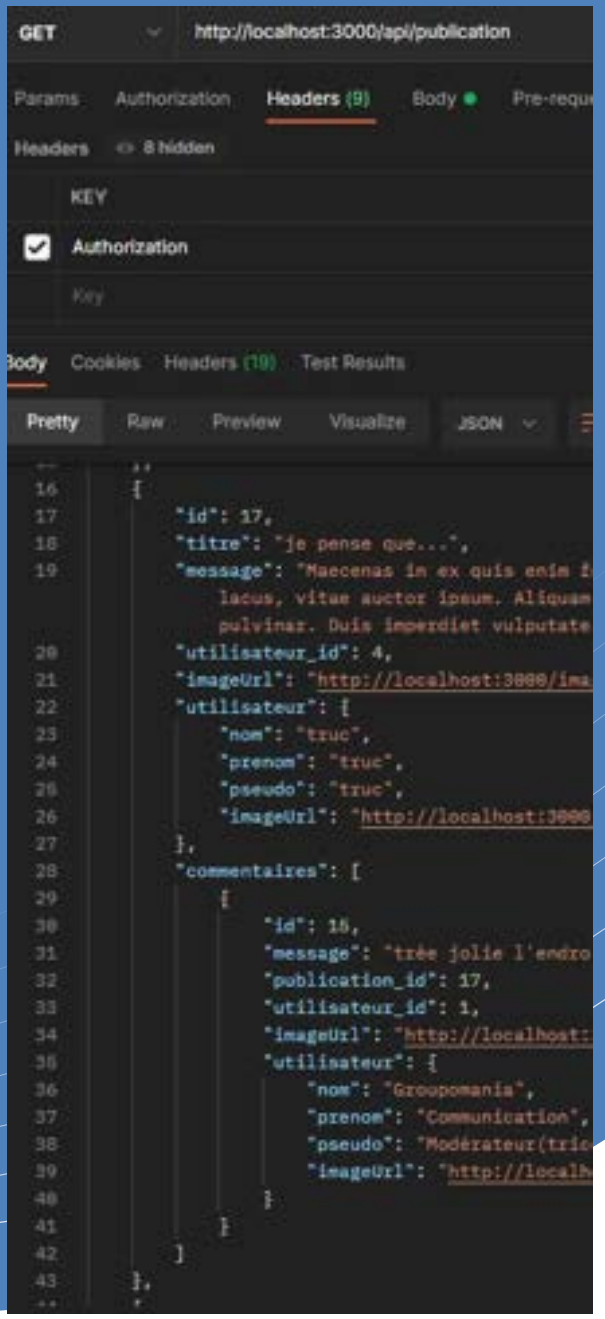
```
async created() {
  let user = localStorage.getItem("user");
  let userLocal = JSON.parse(user);
  await instancePost
    .get("/publication/", {
      headers: {
        Authorization: "Bearer " + userLocal.token,
        "Content-Type": "application/json",
      },
    })
    .then((response) => {
      this.publications = response.data;
      // console.log(this.publications);
    })
    .catch(function (error) {
      alert(error);
      console.log(error);
    });
},
```

Mes requêtes en BDD

- Les requêtes me fournissent toutes les informations associés à mes includes.

```
// recherche d'information all utilisateur
exports.userProfilAll = (req, res, next) => {
  // console.log(_id);
  User.findAll({
    include: [
      { model: Publication, required: false },
      { model: Commentaire, required: false }]
  })
    .then((profil) => res.status(200).json(profil))
    .catch((error) => res.status(505).json({ error }));
};
```

```
exports.getAllCommentaire = (req, res, next) => {
  // findByPk pour trouver tous les objets
  Commentaire.findAll({
    include: [
      { model: Publication, required: true, },
      { model: User, required: true, attributes: ['nom', 'prenom', 'pseudo',
      'imageUrl'] }]
  })
```

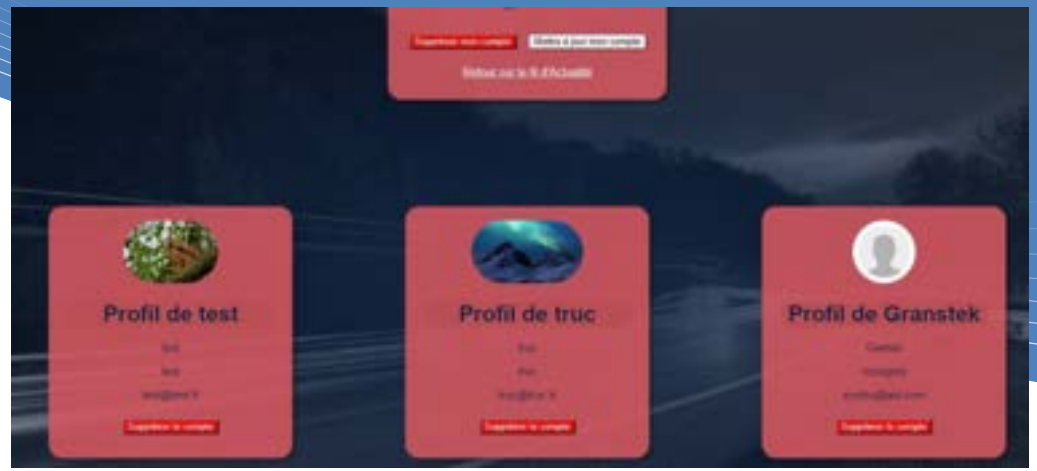


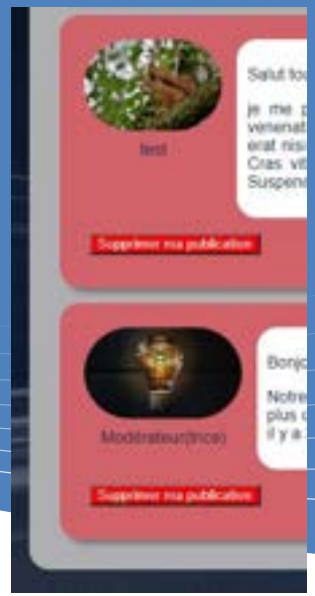
Contrôle administrateur

- J’ai mis en place un compte **ADMIN** avec la possibilité de supprimer n’importe quel compte, publication ou commentaire contrairement au simple compte **utilisateur** qui n’ont accès qu’a leur propre compte, publications et commentaires.

```
admin() {
  let user = localStorage.getItem("user");
  let userLocal = JSON.parse(user);
  if (userLocal.role === "ADMIN") {
    return true;
  } else {
    // console.log(userLocal.role);
    return false;
  }
},
```

- Les profils des utilisateurs sont charger seulement si le compte actuel possède le rôle ADMIN.





```
async created() {
  if (this.admin(true)) {
    let user = localStorage.getItem("user");
    let userLocal = JSON.parse(user);
    await instance
      .get("/profil/", {
        headers: { Authorization: "Bearer " + userLocal.token },
        "Content-Type": "application/json",
      })
      .then((response) => {
        this.profils = response.data;
        console.log(this.profils);
      })
      .catch(function (error) {
        alert(error);
        console.log(error);
      });
  }
},
```

Vue.js 1/2

- Mise en place d'un state qui est un objet unique contenant tous les états de l'application et sert de source unique de vérité que j'ai utilisé pour certains status et les informations utilisateurs.

```
state: {
  status: '',
  user: user,
  userInfos: {
    id: '',
    nom: '',
    prenom: '',
    pseudo: '',
    email: '',
    role: '',
    imageUrl: '',
  },
},
```

- Les mutations me permettent de changer l'état dans le store, elles sont synchrones. Ici nous changeons les informations de profil, les informations de l'utilisateur connecté, le status et la déconnexion.

```
mutations: {
  setStatus(state, status) {
    state.status = status;
  },
  logUser(state, user) {
    instance.defaults.headers.common['Authorization'] = user.token;
    localStorage.setItem('user', JSON.stringify(user));
    state.user = user;
  },
  userInfos(state, userInfos) {
    state.userInfos = userInfos;
  },
  logout(state) {
    state.user = {
      userId: -1,
      token: '',
    }
    localStorage.removeItem('user');
  }
},
```

Vue.js 2/2

- J’utilise ensuite des actions, celle ci par exemple permet de se connecter en effectuant une action asynchrone puis deux demandes de mutation du state.

- On fait l’appel au serveur dans les actions avant de muter l’état de notre application.

```
login: ({ commit }, userInfos) => {
  commit('setStatus', '');
  return new Promise((resolve, reject) => {
    instance.post('/login', userInfos)
      .then(function (response) {
        commit('setStatus', '');
        commit('logUser', response.data);
        resolve(response);
      })
      .catch(function (error) {
        commit('setStatus', 'error_login');
        reject(error);
      });
  });
},
```

Sécurité et contrôle Front-End

- Mise en place d’un contrôle des champs et avertissements visuel destiné à l’utilisateur, remontant le problème en cas de non respect. Valable aussi bien pour l’inscription que la connexion.

```
const regexName = /^[^=*'<>{}0-9]{3,}$/;
const regexMail = /^((?!\.)[\w-_.]*[^.])(@\w+)(\.\w+(\.\w+)?[^.\W])$/;
```

```
signup() {
  this.errorCreate = "";
  this.errorPasswordCreate = "";
  // Récupération des infos pour l'envoie en POST
  // Validation que le formulaire est correctement rempli
  if (
    (regexName.test(this.pseudo) == true) &
    (regexName.test(this.nom) == true) &
    (regexName.test(this.prenom) == true) &
    (regexMail.test(this.email) == true)
  ) {
    const self = this;
    this.$store
      .dispatch("signup", {
        pseudo: this.pseudo,
        prenom: this.prenom,
        nom: this.nom,
        email: this.email,
        password: this.password,
      })
      .then(
        function () {
          self.login();
        },
        function (error) {
          console.log(error);
        }
      );
    this.errorPasswordCreate =
      "Le mot de passe doit contenir au moins 10 caractères, une majuscule, une minuscule, 2
      chiffres, un symbole ainsi qu'aucun espace.";
  } else {
    this.errorCreate = "Email incorrect ou caractère interdit";
    console.log(this.errorCreate);
  }
```