



DOSSIER PROFESSIONNEL (DP)

<i>Nom de naissance</i>	<input type="checkbox"/> DIOUA
<i>Nom d'usage</i>	<input type="checkbox"/> DIOUA
<i>Prénom</i>	<input type="checkbox"/> Samia
<i>Adresse</i>	<input type="checkbox"/> 97 RUE Falguière 75015 PARIS

Titre professionnel visé

Développeur WEB et WEB MOBILE
NIVEAU III

MODALITE D'ACCES :

- ☒ Parcours de formation
- ☐ Validation des Acquis de l'Expérience (VAE)

Présentation

Je suis une femme au foyer et mère d'un enfant avec un diplôme d'ingénieur en réseau et télécommunication que j'ai obtenu au Maroc. Je suis installée en France depuis 2 ans et demi.

Durant mon parcours scolaire, j'ai toujours été passionnée par le code. Continuellement, je cherche à déchiffrer et à réaliser les algorithmes. Je me suis inscrite au pôle emploi d'où j'ai trouvé la formation Konexio que je trouve riche en connaissances techniques et en même temps à jour par rapport aux nouveautés dans le domaine. Ainsi, je me suis lancée dans cette formation pour améliorer mes connaissances afin de faciliter mon intégration au sein de la société.

A court terme, je veux intégrer une équipe très motivée pour apprendre de nouvelles technologies afin d'élargir mes compétences et mettre en pratique mes acquis.

Ce dossier professionnel est le résultat du travail réalisé dans le cadre du projet de fin de formation Digitous. Son objectif est la réalisation d'une application Web qu'on a appelé SharEvent qui permet à l'équipe technico-commerciale de Salesforce de se coordonner pour assister à des événements et gérer la veille technologique.

Ce dossier présente d'une part la réalisation de la partie Backend de l'application web, en citant les étapes que nous avons suivies pour mettre en place une base de données MongoDB et une API CRUD avec NODE-JS.

D'autre part, ce dossier présente la partie Frontend de l'application Web, à savoir l'interface graphique de l'application et l'affichage des données récupérées depuis la base de données. Nous avons réalisé cette partie avec REACT JS, CSS, HTML et Bootstrap.

A la fin de ce rapport, j'ai expliqué la page statique qui présente un landing page de notre application Web. son objectif est de définir notre application comme un produit, à savoir ces caractéristique et ces avantages.



Cahier des charges

L'équipe technique (avant-vente) a besoin de gérer les vieilles technologiques en permanence. Il s'agit de réaliser une application durant deux semaines à compter de 13 Mai au 24 Mai. Cette application permettra à une équipe de se coordonner sur les événements intéressants pour qu'une seule personne de l'équipe y aille et se charge d'ajouter du contenu accessible à toute l'équipe.

Après avoir détaillé le besoin de l'entreprise, on est arrivé à définir les fonctionnalités suivantes :

- Chaque membre de l'équipe a un compte
- L'utilisateur peut ajouter un événement
- L'utilisateur peut ajouter un feedback pour un événement où il a participé
- L'utilisateur peut regarder les événements d'avenir
- L'utilisateur peut avoir une idée sur les feedbacks de tous les utilisateurs
- L'utilisateur peut choisir un événement où il veut participer.

Tâches effectuées

Durant la réalisation de l'application SharEvent, ma mission était la partie Backend en réalisant l'API CRUD avec la base de données MongoDB. Ensuite, je me suis chargée de créer les demandes de données en Frontend avec REACT vers l'API.

Dans ce dossier, je vais présenter une partie du travail réalisé. Je vais l'expliquer en deux parties :

Frontend :

- les étapes que j'ai abordées pour créer le formulaire qui sert à partager un feedback d'un événement pour lequel un membre a participé.
- Les appels API effectués vers le serveur.

Backend :

- La création des schémas et modèles de la base de données,
- La création des routes vers la base de données où les informations vont être stockées.

Table des matières

Présentation	2
Cahier des charges.....	3
Tâches effectuées.....	3
I. Introduction.....	7
II. Définition	7
1. ReactJS.....	7
2. Bootstrap.....	7
III. Création et préparation du projet.....	8
1. Création du projet React	8
2. Routes.....	8
3. Navigation	9
4. Menu	9
IV. Création du formulaire Add-feedback.....	10
1. Action	10
2. Modal	11
3. Composant AddShare.....	12
a. State	12
b. Modifier le state.....	13
c. Sauvegarde feedback	16
d. Hight order Component.....	17
e. Request API.....	18
I. Introduction.....	21
II. Définition	21
1. Mongodb	21
2. Mongoose.....	21
3. NodeJS	21
4. Express.....	21
III. Création et préparation du projet.....	22
1. Création du projet ExpressJS	22

DOSSIER PROFESSIONNEL (DP)

2.	Préparation de la Base de données.....	23
IV.	Routes add feedback	26
I.	Introduction.....	31
II.	Définition	31
1.	Css.....	31
2.	Html	31
3.	jQuery.....	31
III.	Création de la page statique	32
1.	Préparation de la page :	32
2.	Barre de navigation	32
3.	Header de la page.....	33
4.	Section des caractéristiques.....	34
	Annexe.....	37

Partie 1: La réalisation de la partie FrontEnd de l'application Web

I. Introduction

Le but de cette partie est de réaliser le frontend de l'application avec React JS, BOOTSTRAP ET CSS.

Mon développement va suivre les étapes suivantes que je vais ensuite expliquer en prenant des exemples :

1. Création du projet REACT,
2. Définition des routes et création des composants nécessaires,
3. Création de barre de navigation,
4. Le menu de l'application,
5. Développement du formulaire add-feedback,
6. Développement du composant addShare,
7. les appels vers l'API.

II. Définition

1. ReactJS

React est une librairie JavaScript développée par Facebook depuis 2013, qui permet véritablement de révolutionner le développement des interfaces pour les applications web.

La librairie bénéficie d'une grosse communauté et de nombreux modules très utiles sont maintenus dans les adeptes de React pour lui donner encore plus de puissance et de fonctionnalités.

Grâce à React, nous allons pouvoir décomposer nos pages en petits composants indépendants, intelligents et réutilisables, et tout cela avec du simple JavaScript.

Ces composants sont généralement écrits en JSX, un astucieux mélange de JavaScript et d'HTML qui permet de développer les composants de façon claire et intuitive.

2. Bootstrap

Bootstrap est une collection d'outils utiles à la création du design (graphisme, animation et interactions avec la page dans le navigateur, etc.) de sites et d'applications web. C'est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option.

III. Création et préparation du projet

1. Création du projet React

Pour créer le squelette de notre application ShareEvent client de base, j'ai utilisé **create-react-app** qui est un outil pour faciliter le développement d'applications web fondées sur React

L'exécution de la commande peut-être un peu longue car elle déploie toute l'arborescence d'une application React.

```
λ npx create-react-app shareevent
npx : 91 installé(s) en 6.552s

Creating a new React app in C:\Users\Mouni\Desktop\KONEXIO\shareevent\shareevent
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts...
```

Pour démarrer notre application, il faudra lancer la commande **npm start** au sein d'une ligne de commande, depuis le répertoire dans lequel a été créé l'application. Après, l'application de base a dû s'ouvrir automatiquement dans notre navigateur avec **url : http://localhost:5000/**

```
Compiled successfully!

You can now view shareevent in the browser.

Local:            http://localhost:5000/
On Your Network:  http://192.168.1.17:5000/

Note that the development build is not optimized.
To create a production build, use yarn build.
```

2. Routes

Dans le Render de App.js, nous avons importé les composants et nous avons défini les routes de l'application. Comme montre l'image ci-dessous :


```
render() {
  return (
    <div>
      <Router>
        <Route path="/" exact component={LoginContainer} />
        <Route path="/signup" exact component={SignupContainer} />
        <Route path="/home" exact component={HomeContainer} />
        <Route path="/events/:id" component={EventContainer} />
        <Route path="/shared" component={SharedContainer} />
        <Route path="/profile" component={ProfileContainer} />
      </Router>
    </div>
  );
}
```

Parallèlement, Il faut créer les fichiers .js pour tous les containers dans le dossier src/containers/.

3. Navigation

Nous avons défini la barre de navigation de l'App. En appuyant sur le logo, nous nous dirigeons vers la page Home. Le lien profile permet de récupérer les informations sur l'utilisateur depuis la base de données avec la méthode Get. Le lien Logout sert pour que l'utilisateur se déconnecte. Cette barre est accessible dans toutes les pages de navigation.



Profile Logout

4. Menu

Nous avons créé un menu qui permet à l'utilisateur de se diriger vers le contenu qu'il veut sans quitter la page Home. Pour le réaliser, nous avons déclaré le state **activeTab** de type **String** qui prend comme valeur le nom de l'onglet sélectionné.

Ensuite, nous avons créé une méthode qui utilise la méthode react **setState** qui sert à changer le state pour chaque **Click** du tab choisi.

Puis, nous vérifions la sélection avec une condition. Par exemple, si on clique sur shared : activeTab change le state et il prend la valeur shared. Après si la condition True, le composant sharedContainer est appelé pour afficher son contenu.

```

onActiveTab(activeTab){
  this.setState({
    activeTab
  })
}

render() {
  // console.log('Home#props',this.props);
  return(
    <div>
      <Nav />
      <div className="container-fluid">
        <div className="row m-5">
          <div className="col-lg-12">
            <div style={{ marginBottom: 25, textAlign: "center" }}>
              <Button onClick={()=>this.onActiveTab("events")}>
                <i className="fas fa-calendar-alt"></i> Events
              </Button>
              <Button onClick={()=>this.onActiveTab("shared")}>
                <i className="fas fa-share-alt-square"></i> Shared
              </Button>
              <Button onClick={()=>this.onActiveTab("news")}>
                <i className="fas fa-newspaper"></i> News
              </Button>
            </div>
            {this.state.activeTab === "events" && <EventsContainer {...this.props} />}
            {this.state.activeTab === "shared" && <SharedContainer />}
            {this.state.activeTab === "news" && <NewsContainer />}
          </div>
        </div>
      </div>
    </div>
  )
}

```




IV. Création du formulaire Add-feedback


1. Action

Dans cette partie j'ai pris l'exemple d'un utilisateur qui a choisi un évènement où il a participé pour ajouter un feedback. En effet, je vais expliquer les démarches que nous avons suivies pour créer une action. Cette action explique qu'une fois l'utilisateur click sur le bouton **Add feedback**, un modal est ouvert, puis il remplit le formulaire et enfin il ajoute son partage.

Add feedback



2019
FRANCE



19/06/2019 19:00 to 20/06/2019 01:00

28 Rue du Cardinal Lemoine, 75005 Paris

Free

Get your Ticket

```

<div className="container">
  <div className="row">
    <button type="button" className="btn btn-info mb-3" data-toggle="modal" data-target="#ModalFeedback">
      <i className="fas fa-folder-plus"></i> Add feedback
    </button>
  </div>
</div>

```

2. Modal

Dans ce paragraphe, je vais développer la partie du modal. Nous avons utilisé le Framework Bootstrap pour créer un modal avec des Inputs et des labels. Le modal se constitue de son :

Header : dans lequel nous avons affiché le nom d'évènement en cours.

Pour récupérer les informations de chaque évènement, une requête HTTP GET est exécutée par le composant React qui s'appelle **componentDidMount** depuis le composant EventContainer. Cette requête sert à faire un appel synchrone à l'API pour récupérer les informations pour chaque évènement à l'aide de son ID. Ce dernier est récupéré depuis l'URL de la page avec le paramètre **this.props.match.params.id**.

```

async componentDidMount() {
  console.log("this.props", this.props)
  const url = `${Config.HOST_SERVER}/events/${this.props.match.params.id}`
  console.log(url)
  const res = await fetch(url)
  const data = await res.json();
  this.setState({
    informations : data.event
  })
  // console.log(this.state.informations)
}

```

Body : nous avons appelé le composant AddShare. Je vais le détailler dans la partie suivante.

```
<div className="btnBack m-5">
  <Link to="/home" style={{ color: "#4C4B63" }}><i className="fas fa-arrow-circle-left"></i> Back</Link>
</div>
<div className="container">
  <div className="row">
    <button type="button" className="btn btn-info mb-3" data-toggle="modal" data-target="#ModalFeedback">
      <i className="fas fa-folder-plus"></i> Add feedback
    </button>
    <div className="modal fade"
      id="ModalFeedback"
      tabIndex="-1" role="dialog"
      aria-labelledby="ModalLabel"
      aria-hidden="true">
      <div className="modal-dialog modal-lg text-center" role="document">
        <div className="modal-content">
          <div className="modal-header">
            <h5 className="modal-title" id="ModalLabel">{informations.name}</h5>
            <button type="button" className="close" data-dismiss="modal" aria-label="Close">
              <span aria-hidden="true">&times;</span>
            </button>
          </div>
          <div className="modal-body">
            <AddShared {...this.state}/>
          </div>
          <div className="modal-footer">
            <button type="button" className="btn btn-secondary" data-dismiss="modal">Close</button>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Le composant AddShared est importé par la ligne suivante, en définissant son nom et son chemin

```
import AddShared from '../components/shared/AddShared';
```

Footer : on trouve le bouton Close pour fermer le modal.

3. Composant AddShare

a. State

Notre composant a un état (state) dont la valeur par défaut est définie dans getInitialState. Nous n'allons pas, utiliser la méthode getInitialState, car celle-ci est seulement supportée par les classes créées avec React.createClass. Or, nous utilisons une "simple" classe JavaScript. On peut y accéder dans render avec this.state et on l'assigne à la fois en valeur à notre input et en texte.

On applique ce principe pour toutes les états de notre composant. Afin de récupérer les valeurs du champ des inputs du formulaire.

```
class AddShared extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      files: [],  
      contacts: [],  
      idea1: "",  
      idea2: "",  
      idea3: "",  
      name_society: "",  
      name_contact: "",  
      phone: "",  
      summarize: ""  
    };  
  }  
}
```

b. Modifier le state

On utilise l'attribut **onChange** (comme en HTML) sur notre input pour lier l'évènement à une fonction du composant qui a la charge de mettre à jour l'état par la méthode **setState** et c'est là que le virtual-dom entre en jeu. Appeler **setState** va provoquer un rendu (render), dont la valeur de retour va être différente du DOM actuel et les changements seront donc appliqués.

Pour chaque input, nous avons créé une méthode qui met à jour notre state. Ainsi, chaque méthode sera exécutée avec l'attribut **onChange**.


La méthode **onChangeInput()** prend un objet comme paramètre avec deux clefs : **name** récupère le nom de state et **value** récupère la valeur. Cette syntaxe s'appelle le **Destructuring**. Elle est utilisée dans ES6. Prenons l'exemple d'input de type file. Dans l'attribut onChange,

```
onChangeInput({ name, value }) {  
  this.setState({  
    [name]: value  
  });  
}
```

```
<label htmlFor="shareFiles" className="font-weight-bold">  
  Load event files  
</label>  
<input  
  type="file"  
  className="form-control-file"  
  id="shareFiles"  
  multiple  
  onChange={evt =>  
    this.onChangeInput({  
      name: "files",  
      value: evt.target.files  
    })  
  }  
</>
```

Nous exécutons la méthode **onChangeInput**. Cette dernière va récupérer l'ensemble des fichiers chargés par l'utilisateur avec la valeur `evt.target.files` et le nom de state `files`. Puis les stocker dans le state **files** de type **Array**. La même chose pour les autres inputs.

Cependant, le state `Contacts` est géré par une autre façon. C'est un **Array** des objets qui contient trois clés sous format des states (`name_society`, `name_contact`, `phone`). L'utilisateur a le choix d'ajouter le nombre de contacts qu'il a rencontré dans l'évènement.

En cliquant sur , la méthode **createInput()** va retourner le state `contacts` avec un **Array** des objets. Nous avons utilisé la méthode **map()** sur `contacts` pour créer trois inputs avec un key qui identifie chaque ensemble des inputs. En plus, React a toujours besoin de l'attribut **key** pour identifier ces éléments HTML. Cela améliore les performances lors de l'ajout, la modification et la suppression d'un élément.

La méthode **onChangeContact()** s'occupe de mettre à jour le state d'objet de chaque contact en liaison avec son identifiant **index**, puis, elle change le state de `contacts`. Par exemple, si la personne veut modifier le premier objet à l'aide de cet index, la méthode sait quel objet elle va mettre à jour. De même, quand elle veut aussi supprimer un objet avec la méthode `removeClick()`. Cette dernière, elle va prendre comme paramètre l'index d'objet dans l'**Array** ensuite elle va le supprimer avec la méthode **splice**.

DOSSIER PROFESSIONNEL (DP)

```

createInput() {
  return this.state.contacts.map((contact, i) => (
    <div key={i}>
      <input
        type="text"
        className="form-control m-2"
        placeholder="Company Name"
        value={contact.name_society}
        onChange={evt =>
          this.onChangeContact(
            { name: "name_society", value: evt.target.value },
            i
          )
        }
      />
      <input
        type="text"
        className="form-control m-2"
        placeholder="Contact"
        value={contact.name_contact}
        onChange={evt =>
          this.onChangeContact(
            { name: "name_contact", value: evt.target.value },
            i
          )
        }
      />
      <input
        type="text"
        className="form-control m-2"
        placeholder="Phone Number"
        value={contact.phone}
        onChange={evt =>
          this.onChangeContact({ name: "phone", value: evt.target.value }, i)
        }
      />
      <a
        style={{ cursor: "pointer" }}
        onClick={this.removeClick.bind(this, i)}
      >
        <i className="fas fa-trash-alt" />
      </a>
    </div>
  ));
}

```

```

onChangeContact({ name, value }, index) {
  const newContacts = this.state.contacts;
  newContacts[index][name] = value;
  this.setState({
    contacts: newContacts
  });
}

```

```

removeClick(i) {
  let contacts = [...this.state.contacts];
  contacts.splice(i, 1);
  this.setState({ contacts });
}

```

En même temps **addInput()** s'exécute pour stocker dans l'Array contacts les objets avec la méthode **push**. Comme illustre le code ci-dessous


```
onAddInput() {  
  const input = {  
    name_society: this.state.name_society,  
    name_contact: this.state.name_contact,  
    phone: this.state.phone  
  };  
  const newInput = this.state.contacts;  
  newInput.push(input);  
  this.setState({  
    contacts: newInput  
  });  
}
```

c. Sauvegarde feedback

Après avoir traité toute la logique pour récupérer les valeurs des inputs et les stocker dans des states. Maintenant, on arrive à la phase de sauvegarde, je vais expliquer le rôle de la méthode **saveShare()**.

Nous avons utilisé pour envoyer les données du formulaire à l'API **FormData**. L'objet **FormData** fourni un moyen facile pour construire un ensemble de paires clé / valeur qui représentent les champs du formulaire et leurs valeurs, qui peuvent ensuite être facilement envoyés à l'aide de la méthode **POST**. La méthode **append()** ajoute une paire clé / valeur à l'objet **FormData**.

Ensuite, nous avons appelé la méthode **postShare()** qui est créée dans le fichier **API.js**. Elle prend comme paramètre **user** connecter (la récupération d'utilisateur authentifié dans la partie

suivante), objet **data** qui contient toutes les données des inputs et **id** d'évènement où l'utilisateur a participé. Une fois que la requête est envoyée, l'utilisateur se dirige vers la liste des shares.

```
saveShare() {
  const { files, contacts, idea1, idea2, idea3, summarize } = this.state;
  const { user } = this.props.user;
  const { _id } = this.props.information;
  let data = new FormData();
  for (const file of files) {
    data.append("files", file, file.name);
  }
  data.append("main_ideas", JSON.stringify({ idea1, idea2, idea3 }));
  data.append("contacts", JSON.stringify(contacts));
  data.append("summarize", summarize);
  Api.postShare(data, user, _id).then(res => {
    console.log("res", res);
    if (res.error) {
      alert(res.error);
    } else {
      this.props.history.push("/shared");
    }
  });
}
```

```
<button
  data-dismiss="modal"
  type="button"
  className="btn btn-info"
  onClick={() => this.saveShare()}
>
  Add
</button>
```

d. Higher Order Component

Higher Order Component (ou HOC). Ce qui ça veut dire, qu'on va simplement wrapper un composant dans un autre. Lequel a pour unique rôle d'injecter la fonctionnalité et de la passer via les props. Il s'agit du principe de composition : au lieu d'exporter A, on exporte Wrapped(A), et ce dernier retourne un composant React qui va appeler A dans sa méthode render.

Pour récupérer à tout moment l'utilisateur authentifié, nous avons réalisé un composant considéré comme un HOC **withUser** utilisable dans toute l'application.



withUser permet de créer une fonction qui retourne un composant. Ce dernier récupère les informations de l'utilisateur avec la méthode **getUser()** que nous avons créée dans le fichier API.js. Après, il vérifie si l'utilisateur est authentifié avec la méthode **isAuthenticated**, afin de passer les informations de l'utilisateur dans le composant.

Nous avons utilisé aussi le withRouter de react-router-dom. withRouter est un composant d'ordre supérieur de React Router qui permet le rendu de son composant chaque fois que l'itinéraire change avec les mêmes props. **history.push** est un moyen de rediriger les utilisateurs à la page souhaitée après la validation.

```
import React from 'react';
import Api from '../utils/Api';

const withUser = (WrappedComponent) => {
  return class extends React.Component {
    render() {
      const user = Api.getUser();
      // console.log(user)
      user.isAuthenticated = () =>
        user.hasOwnProperty('_id') && user._id !== null;
      return <WrappedComponent user={user} {...this.props} />
    }
  }
};

export default withUser;
```

```
import React from "react";
import Api from "../../utils/Api";
import { withUser } from "../../user";
import { withRouter } from "react-router-dom";
```

```
export default withRouter(withUser(AddShared));
```

e. Request API

J'ai déjà abordé ci-dessus la méthode postShare() utilisée dans la méthode saveShare(). Ici, je vais expliquer son rôle. Donc postShare prend dans ces paramètres les informations du formulaire, le user et event id . Elle sert à faire un appelle à l'API de type POST via l'URL : <http://localhost:3002/api/events/id-Event/users/id-user/shares> avec la méthode Fetch.

L'**API Fetch** fournit une interface JavaScript pour l'accès et la manipulation des parties de la pipeline HTTP, comme les requêtes et les réponses. Cela fournit aussi une méthode globale **fetch()** qui procure un moyen facile et logique pour récupérer des ressources à travers le réseau de manière asynchrone.

```

postShare(share, user, event) {
  const options = {
    method: "POST",
    body: share,
    headers: {
      "Content-Type":
        "multipart/form-data; boundary=-----021198133891922950270909"
    }
  };
  delete options.headers["Content-Type"];
  return fetch(
    `${Config.HOST_SERVER}/events/${event}/users/${user._id}/shares`,
    options
  )
    .then(res => res.json())
    .then(json => json);
}

```

Le résultat du développement du composant addShare est présenté avec les captures d'écrans ci-dessous. Cela termine ma présentation d'exemple de développement de la partie FRONT-END de l'application web shareEvent.

Soirée des Trophées ChooseMyCompany - HappyIndex® AtWork 2019

×

Load event files

Sélect. fichiers

Aucun fichier choisi

Give 3 Main Ideas

First idea

Second idea

Third idea

Person or Company encountered

Company Name

Contact

Phone Number

🗑️

+

Summarize

Add

Close

Partie 2: La réalisation de la partie BackEnd de l'application Web

I. Introduction

L'objectif de cette partie est d'expliquer la partie backend de l'application avec expressJs et mongoDB.

Mon développement va suivre les étapes suivantes :

1. Création du projet expressJs
2. Préparation de la base de données
3. Les collections de l'application
4. Créations des routes en prenant l'exemple de Add feedback

II. Définition

1. MongoDB

MongoDB qui est un système de gestion de base de données NOSQL donc non relationnelles, suffisante pour le travail à effectuer. A la différence des bases de données relationnelles qui s'organisent sous forme de tables, MongoDB et les bases NOSQL s'organisent sous forme de documents (des collections) constitués d'un ensemble de paires clés-valeurs, formatés au format JSON.

MongoDB est libre de droits et facile à installer. Il dispose d'un langage d'interrogation qui lui est propre.

2. Mongoose

Mongoose qui est un module Node.js qui s'installe avec npm. Il permet de faire la passerelle entre le serveur Node.js et le serveur MongoDB.

3. NodeJS

NodeJS est une plateforme construite sur le moteur JavaScript de Chrome qui permet de développer des applications en utilisant du JavaScript. Il se distingue des autres plateformes grâce à une approche non bloquante permettant d'effectuer des entrées/sorties (I/O) de manière asynchrone.

4. Expresss

Express.js est un framework pour Node.js. Il nous fournit des outils de base pour aller plus vite dans la création d'applications Node.js. Il permet de créer un simple serveur de fichiers statiques comme un serveur d'API en passant par la création d'un site dynamique.

III. Création et préparation du projet

1. Création du projet ExpressJS

Pour créer notre projet SharEvent côté serveur, nous avons commencé par créer un dossier sharevent_db et après procéder à l'initialisation du package **npm** avec la commande **npm init** et à l'installation d'**Express.js** avec la commande **npm -i express**.

```
C:\Users\Mouni\Desktop\sharevent_db
λ npm init
This utility will walk you through creating a package.json file.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (sharevent_db)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\Mouni\Desktop\sharevent_db\package.json:
{
  "name": "sharevent_db",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Nous importons le module express après installation avec la ligne suivante :

```
const express = require("express").
```

Après, nous faisons l'instanciation avec `const app = express()`. Puis, nous utilisons la méthode **listen** pour que l'application démarre un serveur et écoute le port 3002 à la recherche de connexions.

```
app.listen(port, function() {
  console.log("Server started on port:", port);
});
```

Nous installons aussi le module npm **mongoose**. il va servir de passerelle entre notre serveur Node.js et notre serveur MongoDB.

```
const mongoose = require("mongoose");
```

Ensuite nous devons préciser sur quelle base de données nous allons travailler :

PORT=3002

HOST=localhost

DB_NAME=sharevent

DB_PORT=27017

DB_HOST=localhost

Ici la base de données se nomme «sharevent». La méthode connect() va créer un objet Connection accessible via mongoose.connection.

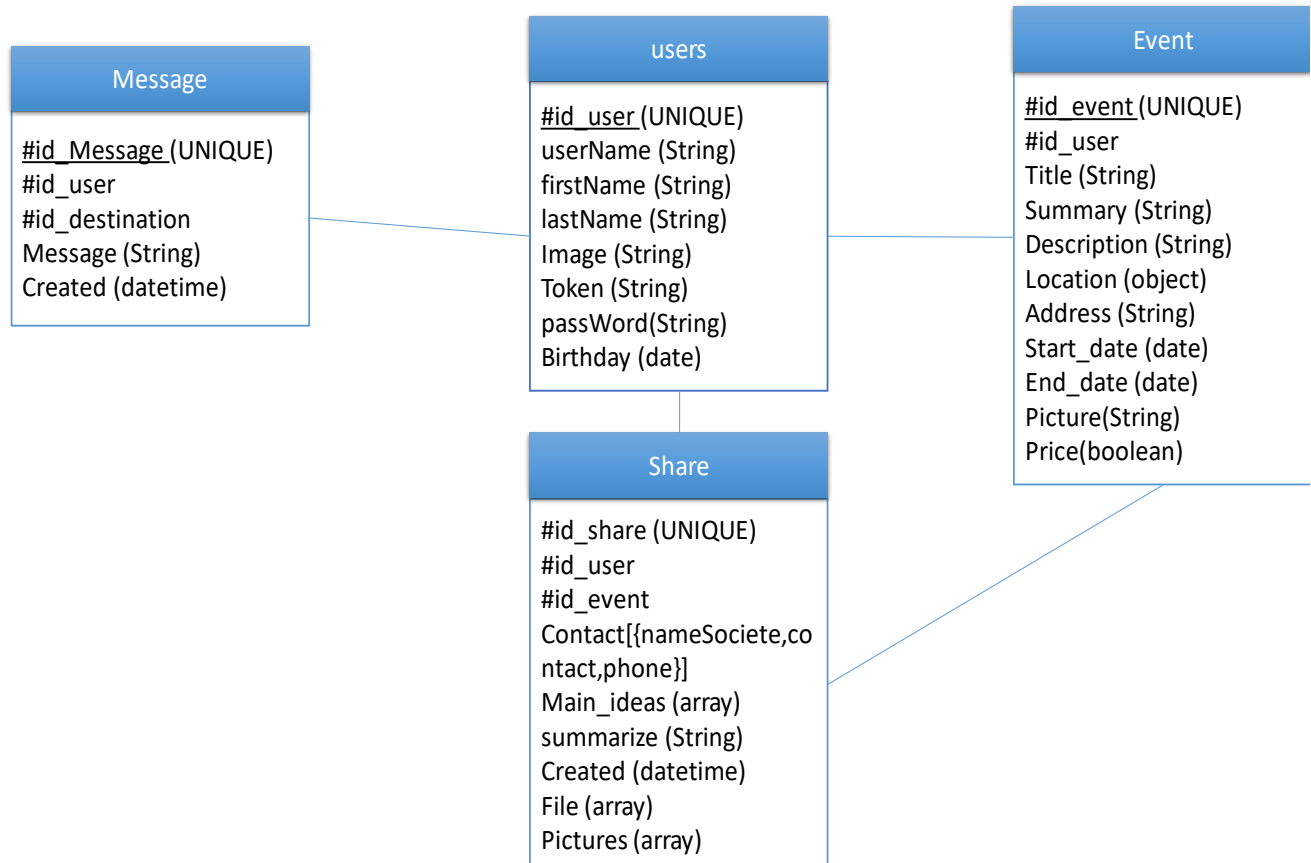
```
const mongoUri =  
  process.env.MONGODB_URI || `mongodb://${DB_HOST}:${DB_PORT}/${DB_NAME}`;  
mongoose.connect(mongoUri, {  
  useNewUrlParser: true,  
  useCreateIndex: true  
});
```

Pour démarrer notre projet, il faudra lancer la commande **node index.js** au sein d'une ligne de commande, depuis le répertoire dans lequel a été créé l'application. Nous devons aussi lancer la base de données avec la commande **mongod -dbpath=./db**.

```
λ node index  
Server started on port: 3002
```

2. Préparation de la Base de données

Après avoir étudié les collections que nous avons besoin pour créer notre application, nous avons mis en place le modèle des données avec la représentation suivante :



Ici, je vais expliquer le schéma et modes de la collection share. La collection share a des relation one to many avec la collection user et event. Elle va prendre comme clé étranger id de la collection user et id de la collection event

La collection Shares pour recenser les partages avec l'utilisateur qu'il a partagé le feedback, event où il a participé, les fichiers, les personnes rencontrées, les points principaux retenus, le résumé.

Mongoose utilise des « Schéma » pour modéliser les données. Il permet de définir les types de variables et de structurer les données comme illustre le code ci-dessous.


```
let mongoose=require('mongoose')
module.exports = {
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User"
  },
  event: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "Event"
  },
  files:[String],
  contacts:[Object],
  main_ideas:Object,
  summarize:String,
  created: {
    type: Date,
    default: Date.now
  }
}
```

Maintenant que nous avons notre schéma de définition, nous allons créer un model qui va nous permettre d'insérer des données dans MongoDB en respectant le schéma précisé et d'aller faire des requêtes dessus !

`let Share = mongoose.model("Share", Schema):` Cela va créer un model nommé **Share** à partir du **schema** et qui va le renvoyer dans la variable Share.

```
let mongoose=require('mongoose');

let schema=require('./schemas').Share;

let Schema=new mongoose.Schema(schema);

let Share = mongoose.model("Share", Schema);

module.exports=Share;
```

IV. Routes add feedback

Après avoir déterminé les bases pour créer une API, passons maintenant à la création des routes. On définit la route `"/api/events/:id/users/:idUser/shares"` qui sert à faire la demande avec la méthode POST vers la base de données,

Nous importons le model `const ShareModel = require("./models").Share`. Après nous créons une instance de Model. Puis, nous récupérons la valeur de chaque fields avec `req.body` et `req.params` si on parle de la récupération des informations depuis url.

```
const share = new ShareModel({
  description: req.body.description || "",
  files: documents || "",
  user: req.params.idUser,
  event: req.params.id,
  contacts: JSON.parse(req.body.contacts),
  main_ideas: JSON.parse(req.body.main_ideas),
  summarize: req.body.summarize || ""
});
```

Cependant, les données ne sont pas encore sauvegardées dans MongoDB. Pour cela, il suffit d'appeler la méthode `save()` de l'instance créée. Cette fonction prend une fonction qu'elle appellera après l'insertion dans la base en donnant comme paramètre l'erreur s'il y en a une et le deuxième paramètre `share` où se trouvent les données insérées sous forme JSON.

```
share.save(function(err, share) {  
  if (err !== null) {  
    return res.json({  
      message: "Error when saving share.",  
      error: err  
    });  
  }  
  return res.json({  
    success: true,  
    data: share  
  });  
});
```

Parmi les fields de la collection nous avons files. Il sert de charger des fichiers. Nous utilisons le module **Multer** pour la gestion **multipart/form-data**, principalement utilisé pour l'upload du fichiers. Puis, nous utilisons le module **fs** pour sauvegarder le fichier dans le dossier public/uploads/fileshare.

```
app.post(  
  "/api/events/:id/users/:idUser/shares",  
  uploadFiles.fields([ { name: "files" } ]),  
  function(req, res) {  
    let documents = [];  
    let files = req.files || {};  
    console.log("res.body", res.body);  
    if (Object.keys(files).length !== 0) {  
      documents = files["files"].map(  
        file =>  
        `uploads/fileshare/${file.filename}.${file.mimetype}`  
      );  
      files["files"].map(file => {  
        return fs.rename(  
          file.path,  
          `public/uploads/fileshare/${file.filename}.${file.mimetype}`  
        ),  
        function() {  
          console.log("file has been uploading!");  
        }  
      });  
    }  
  });
```

```
const fs = require("fs");  
const multer = require("multer");  
const uploadFiles = multer({ dest: "public/uploads/fileshare" });  
const uploadEvents = multer({ dest: "public/uploads/events" });
```

Maintenant, nous allons voir la base de données sharevent dans MongoDB. Nous ouvrons le terminal et nous lançons la commande **mongo. Db.shares.find** récupère tous les documents de la collection shares.

```
> use sharevent
switched to db sharevent
> show collections
events
shares
users
> db.shares.find()
{ "_id" : ObjectId("5cf16994f91934131c95b848"), "files" : [ ],
7b3667c91bc1e"), "main ideas" : { "idea1" : "", "idea2" : "", "
```

Prenons l'exemple de récupération d'un share. Nous utilisons la méthode **Get** avec la requête qui envoie à la base de données **findOne**. Cette dernière retourne l'objet de l'id précisé dans url et la requête. Nous utilisons aussi la méthode **populate()** qui permet de référencer des documents dans d'autres collections. Donc, à l'aide de populate nous pouvons extraire les détails de l'évènement juste avec son id et la même chose avec l'user.

DOSSIER PROFESSIONNEL (DP)

```
app.get("/api/shares/:id", function(req, res) {  
  ShareModel.findOne({ _id: req.params.id })  
    .populate("user")  
    .populate("event")  
    .exec(function(err, share) {  
      if (err) {  
        return res.json({  
          message: "Error when getting share.",  
          error: err  
        });  
      }  
      return res.json({  
        success: true,  
        data: share  
      });  
    });  
});
```

Partie 2: La réalisation d'une page statique

I. Introduction

Dans cette présente partie, je vais expliquer la page statique qu'on a réalisé pour identifier notre application. Pour réaliser cette page, nous avons utilisé CSS, HTML et jQuery.

Notre page est devisée en quatre parties :

- la barre de navigation
- header
- section des caractéristiques de l'application
- section des étapes pour naviguer dans l'application

Je vais expliquer dans les parties suivantes la barre de navigation, le header et la section des caractéristiques.

II. Définition

1. Css

Le CSS, est le diminutif de Cascading StyleSheets. Le CSS a été créé en 1996 et a pour rôle de mettre en forme du contenu en lui appliquant ce qu'on appelle des styles. Ce langage va nous permettre par exemple de définir la taille, la couleur ou l'alignement d'un texte.

Nous allons donc utiliser le CSS sur notre code HTML, afin d'enjoliver le résultat visuel final.

2. Html

HTML est l'abréviation de HyperText Markup Language. Ce langage a été créé en 1991 et a pour fonction de structurer et de donner du sens à du contenu.

Grâce au HTML, on va par exemple pouvoir indiquer au navigateur que tel texte doit être considéré comme un simple paragraphe ou que tel autre est un titre.

3. jQuery

jQuery est un framework Javascript sous licence libre qui permet de faciliter des fonctionnalités communes de Javascript.

L'utilisation de cette bibliothèque permet de gagner du temps de développement lors de l'interaction sur le code HTML d'une page web, l'AJAX ou la gestion des événements. jQuery possède par la même occasion l'avantage d'être utilisable sur plusieurs navigateurs web (cf. Internet Explorer, Firefox, Chrome, Safari ou Opera).

La bibliothèque jQuery possède les fonctionnalités suivantes :

- Manipulation du DOM (HTML ou CSS),
- Gestion des événements (clic, survol, soumettre un formulaire ...),
- AJAX,
- Effet d'animation.

III. Création de la page statique

1. Préparation de la page :

A chaque démarrage pour créer une page statique, nous devons importer tous les fichiers et les liens importants dans la balise **<head>**.

```
<head>
  <title>
    | SharEvent App
  </title>
  <meta charset='utf8'>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css"
    | integrity="sha384-GJzZqFGwb1QTTN6wy59ffF1BuGJpLSa9DkKMP0DgiMDm4iYm70gZWKYbI706tWS" crossorigin="anonymous">
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.8.2/css/all.css"
    | integrity="sha384-oS3vJWv+0UjzBfQzYUhtDYw+Pj2yciDJxpsK10YPAYjqT085Qq/1cq5FLXAZQ7Ay" crossorigin="anonymous">
  <link rel="stylesheet" href="css/all.min.css" />
  <link rel="stylesheet" href="css/animate.css" />
  <link href="css/font-awesome.min.css" rel="stylesheet">
  <link rel="stylesheet" href="css/index.css" />
</head>
```

En plus, nous devons importer les scripts javascript utilisés, de préférence les mettre avant la fermeture de la balise **<body>**.

```
<!-- SCRIPT JAVASCRIPT -->

<script src="js/jquery-3.3.1.min.js"></script>
<script src="js/smooth-scroll.polyfills.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js"
  | integrity="sha384-wHAiFfRlMFy6i5SRaxvfOCifBUQy1xHdJ/yoi7FRNXMRBu5WHdZYu1hA6ZObLgut" crossorigin="anonymous">
</script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js"
  | integrity="sha384-B0UglyR+jN6CkvvICOB2joaf5I4l3gm9GU6Hc1og6Ls7i6U/mkkaduKaBhlAXv9k" crossorigin="anonymous">
</script>
<script src="js/wow.min.js"></script>
<script src="js/main.js"></script>

</body>
```

2. Barre de navigation

Pour créer la navbar, nous nous sommes basés sur le Framework Bootstrap. pour l'utiliser, nous l'avons importé d'une manière locale en téléchargeant le fichier **all.min.css**.


```
<!-- navbar -->
<nav class="navbar navbar-b navbar-light bg-light navbar-expand-md fixed-top navbar-reduce">
  <a class="navbar-brand" href="#"></a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
    aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
    <div class="navbar-nav">
      <a class="nav-item nav-link js-scroll" href="#carouselIndicators">Home <span
        class="sr-only">(current)</span></a>
      <a class="nav-item nav-link js-scroll" href="#steps">Steps</a>
      <a class="nav-item nav-link js-scroll" href="#feature">Features</a>
    </div>
  </div>
</nav>
```

En cliquant sur Chaque lien de la barre nous dirigeons l'utilisateur vers la partie souhaitée en scrollant. L'attribut href prend l'id de chaque section comme référence en utilisant **#lenomdel'id** de la section. Ensuite, nous avons une partie d'animation pour le scroll dans le fichier main.js.

```
$(function() {
  $(".js-scroll").on('click', function(event) {
    if (this.hash !== "") {
      event.preventDefault();
      var hash = this.hash;
      $('html, body').animate({
        scrollTop: $(hash).offset().top
      }, 800, function(){
        window.location.hash = hash;
      });
    }
  });
});
```

3. Header de la page

Ici, nous avons créé un carrousel pour des image des évènements, une texte présentative de l'application et un capture d'écran de la page principale de l'application.

Pour réaliser le carrousel, nous avons utilisé Bootstrap avec son fichier **animate.css**.

```
<!-- header -->
<header>
  <div id="carouselIndicators" class=" carousel slide" data-ride="carousel">
    <div class="carousel-inner">
      <div class="carousel-item image active">
        
      </div>
      <div class="carousel-item image">
        
      </div>
      <div class="carousel-item image">
        
      </div>
      <div class="carousel-item image">
        
      </div>
      <div class="carousel-caption header-desc">
        <div class='row '>
          <div class='col-6 col-md-6 col-lg-4 text'>
            <h3>SharEvent</h3>
            <p>The Best App you will find to get your team up-to-date,
              organized and connected one to others.
            </p>
          </div>
          <div class='col-6 col-md-6 col-lg-4 offset-lg-4'>
            
          </div>
        </div>
      </div>
    </div>
  </div>
</header>
```

4. Section des caractéristiques

Bootstrap se base sur le principe des grilles pour présenter et deviser la page HTML . Une grille est découpée en rangées (appelées **row**, parce que tout est en anglais) et colonnes (**col**). Il faut ensuite définir le nombre de colonnes pour chaque élément en sachant qu'il y en a au maximum 12.

Bootstrap considère 5 sortes de médias :

- les très petits, de type smartphones en mode portrait (moins de 576px),
- les petits, de type smartphones en mode paysage (plus de 576px mais moins de 768px),
- les moyens, de type tablettes (plus de 768px mais moins de 992px),
- les grands (plus de 992px mais moins de 1200px),
- les très grands (plus de 1200px).

DOSSIER PROFESSIONNEL (DP)

```
<div class="container-fluid">
  <div class="row">
    <section class="col-12 col-md-12 col-lg-12 features" id="feature">
      <div class="section-title">
        <h2>Features</h2>
      </div>
    </div>
    <div class="row">
      <div class="col-12 col-md-4 col-lg-2">
        <div class="box-feature wow fadeInUp" data-wow-delay="200ms">
          <h4><i class="fas fa-lock"></i> Security</h4>
          <p>Access to the data is secured.
            Data access is granted only to the authorized person.
          </p>
        </div>
      </div>
      <div class="col-12 col-md-4 col-lg-2">
        <div class="box-feature wow fadeInUp" data-wow-delay="400ms">
          <h4><i class="fas fa-arrow-circle-up"></i> Convenience</h4>
          <p>Easy access to the calendar of the event.
            Get a quick view on all the upcoming events.
          </p>
        </div>
      </div>
      <div class="col-12 col-md-4 col-lg-2">
        <div class="box-feature wow fadeInUp" data-wow-delay="600ms">
          <h4><i class="fas fa-calendar-alt"></i> Organize</h4>
          <p>Coordinate the attendance of team members to an event.
            Attendance to the event will be display ed on the calendar.
          </p>
        </div>
      </div>
      <div class="col-12 col-md-4 col-lg-2">
        <div class="box-feature wow fadeInUp" data-wow-delay="800ms">
          <h4><i class="fas fa-share-alt-square"></i> Share</h4>
          <p>Share easily files with your team.
        </p>
        </div>
      </div>
    </div>
  </div>
</div>
```

Nous avons aussi ajouté le style par exemple pour chaque feature dans le fichier style.css. en ajoutant la bordure, des transitions **:hover**, et la taille de chaque box.

```
background-color: #fff;
max-height: 230px;
padding: 40px 25px;
margin-top: 50px;
border-radius: 10px;
border-bottom: 3px solid #717171;
border-left: 3px solid #717171;
border-right: 3px solid #717171;
border-top: 3px solid #717171;
box-shadow: 0px 7px 50px 0px rgba(0, 0, 0, .1);

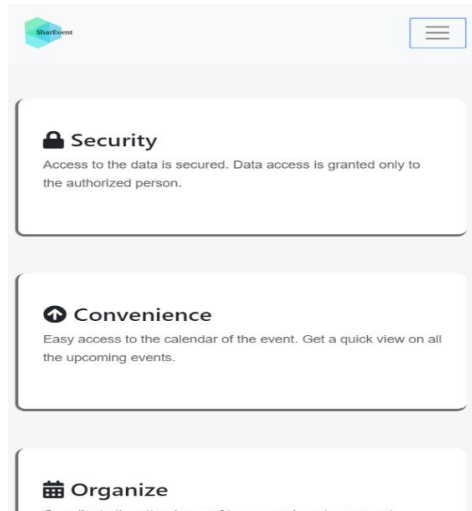
.box-feature:hover {
  transform: translateY(-10px);
  -webkit-transform: translateY(-10px);
  -moz-transform: translateY(-10px);
  transition: transform 0.4s ease;
  -webkit-transition: transform 0.4s ease;
  -moz-transition: transform 0.4s ease;
}

.box-feature h3 {
  display: inline-block;
  padding-left: 50px;
  padding-top: 3px;
  padding-bottom: 3px;
  color: #616161;
  font-family: 'Open Sans', sans-serif;
  font-weight: 600;
  font-size: 20px;
  letter-spacing: 1px;
  text-transform: capitalize;
}
```

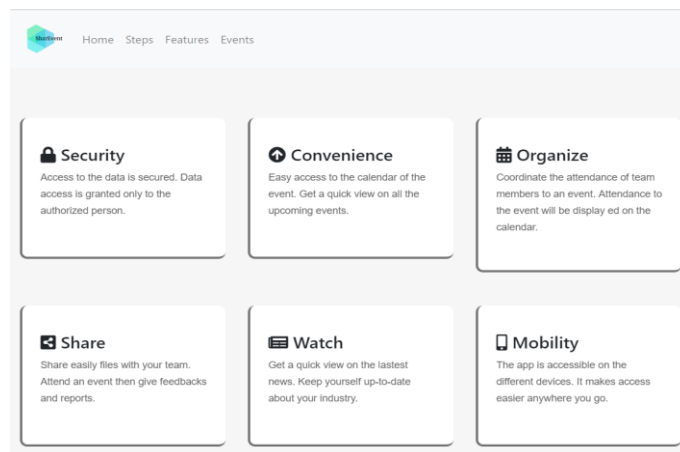
Dans notre exemple, nous avons six caractéristiques qui seront adaptées pour chaque version :

DOSSIER PROFESSIONNEL (DP)

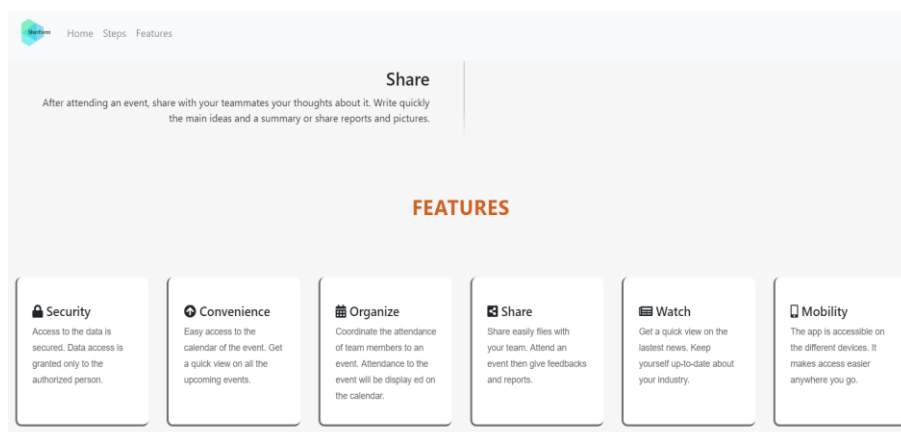
- Mobile col-12,



- Tablette col-md-4,



- Ordinateur col-lg-2.



Annexe

1. <https://www.w3schools.com>
2. <https://stackoverflow.com/>
3. <https://reactjs.org/>
4. <https://www.npmjs.com/>
5. <https://mongoosejs.com/docs/>
6. <https://expressjs.com/en/guide/routing.html>
7. <https://getbootstrap.com/docs/4.3/getting-started/introduction/>

1. les moyens utilisés :

Node.Js
Express.js
NPM
MongoDB
ReactJS
Mongoose
Bootstrap
Html
Css
Jquery

2. avec qui j'ai travaillé :

Salim OKAR
Gautier BLONDEL
Yacine SOUIKI

4. Contexte

Nom de l'entreprise ► *Salesforce*

Projet	► Application web SharEvent projet de fin de formation
Période d'exercice	► Du 13/05/2019 au 24/05/2019

DOSSIER PROFESSIONNEL (DP)

Titres, diplômes, CQP, attestations de formation

Intitulé	Autorité ou organisme	Date
Ingénieure en réseaux et télécommunications	Ecole Nationale du Science Applique	2014
Licence Professionnelle en Électronique, Électrotechnique, Automatique et Réseaux	Faculté Polydisciplinaire	2012
Baccalauréat Scientifique	Lycée Youssef Ben Tachfine	2009

Déclaration sur l'honneur

Je soussignée Samia DIOUA, déclare sur l'honneur que les renseignements fournis dans ce dossier sont exacts et que je suis l'auteur(e) des réalisations jointes.

Fait à Paris le 03/06 /2019

pour faire valoir ce que de droit.

Signature :

Samia DIOUA

