



# CRYPTONEWBIE

## Rapport Mi-Mandat

C61

Lundi 02 Mai 2022

Ouimet Nikolas, Sanon Yvanoski

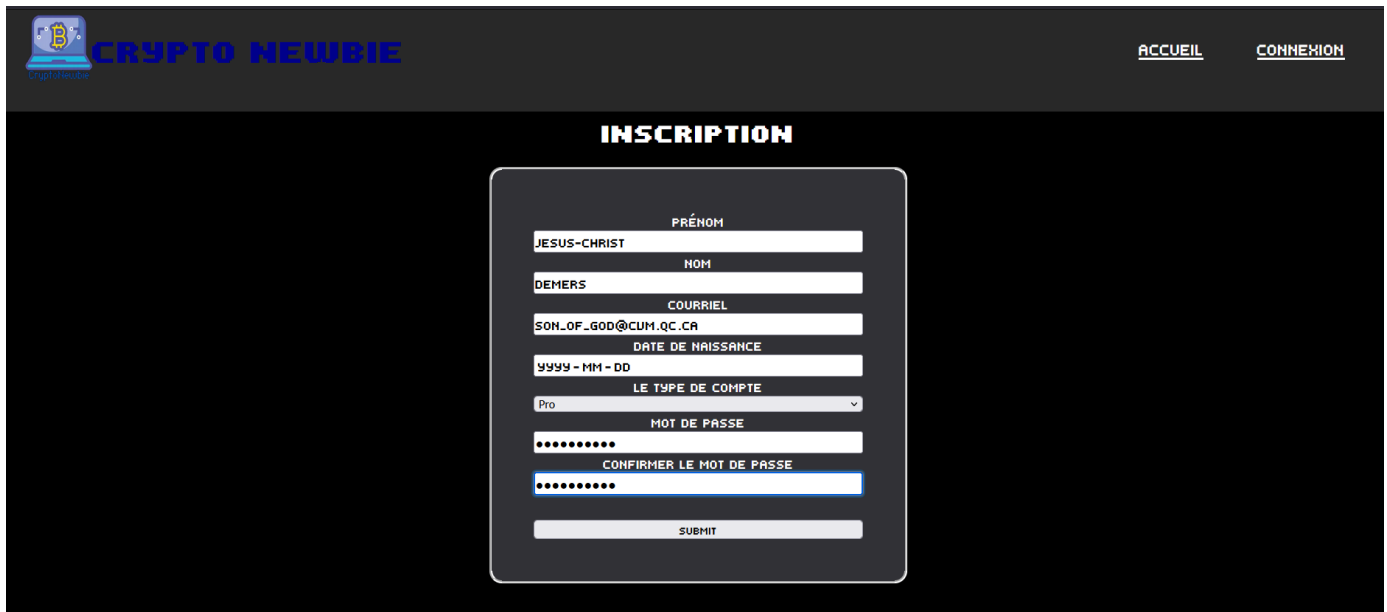
## Yvanoski

Une des fonctionnalités réalisées par Yvanoski est la configuration et l'implémentation de la base de données, ce qui inclus l'inscription et la connexion au site d'un usager.

Le plus grand défi fut de comprendre la documentation pour le module utilisée. En effet, nous faisons l'utilisation d'un ORM (Object-Relational Mapping) pour gérer nos requêtes SQL et utiliser ce genre de module pour la première fois fut plus complexe que prévu au début.

Le code pour cette fonctionnalité se retrouve à plusieurs endroits :

- Les entities (tables) se trouvent sous : `/backend/entity/*`
- Le code pour la connexion à la BD sous : `/backend/BDutils.ts`
- Le code pour l'inscription au site sous : `/pages/api/signup.ts`
- Le code pour la connexion au site sous : `/pages/api/login.ts`



The screenshot shows the 'CRYPTO NEWBIE' website with a dark theme. The header includes a Bitcoin logo and the site name 'CRYPTO NEWBIE' on the left, and navigation links 'ACCUEIL' and 'CONNEXION' on the right. The main content area is titled 'INSCRIPTION' and contains a registration form. The form fields are: 'PRÉNOM' (filled with 'JESUS-CHRIST'), 'NOM' (filled with 'DEMERS'), 'COURRIEL' (filled with 'SON\_OF\_GOD@CUM.QC.CA'), 'DATE DE NAISSANCE' (placeholder 'YYYY-MM-DD'), 'LE TYPE DE COMPTE' (dropdown menu with 'Pro' selected), 'MOT DE PASSE' (masked with dots), 'CONFIRMER LE MOT DE PASSE' (masked with dots), and a 'SUBMIT' button at the bottom.

Exemple d'inscription au site (Front end par Nikolas)

```

export default function signup(
  req: NextApiRequest,
  res: NextApiResponse<{status:string, errors:string[]}>
) {
  utils.getConnection().then(async connection => {

    // Arguments
    const { lastName, firstName, email, accountType, password, confPassword, dateOfBirth } = req.body // arguments reçu du form dans signupForm.tsx
    if (password !== confPassword){
      res.status(400).json( {status:"erreur", errors:["Les deux mots de passes ne sont pas identiques"]})
      return
    }
    //const nomListe = req.body // arguments reçu de création de liste
    const listName = "Première liste"

    // Classes
    const user = new User(lastName, firstName, email, AccountType[accountType].Type, AccountType[accountType].Amount, password, dateOfBirth)
    const portfolio = new Portfolio()
    const transaction = new Transactions()
    const lists = new PersoList(listName)
    const crypto = new Crypto("doge", 10) // Cadeau de bienvenue

    // Repos
    const cryptoRepo = connection.manager.getRepository(Crypto)
    const listsRepo = connection.manager.getRepository(PersoList)
    const transactionRepo = connection.manager.getRepository(Transactions)
    const portfolioRepo = connection.manager.getRepository(Portfolio)
    const userRepo = connection.manager.getRepository(User)

    // Liens entre tables
    lists.crypto = [crypto]
    portfolio.value = AccountType[accountType].Amount
    portfolio.perso_list = [lists]
    user.transactions = [transaction]
    user.portfolio = portfolio

    // Sauvegarde des entities(table)
    await cryptoRepo.save(crypto);
    await listsRepo.save(lists);
    await transactionRepo.save(transaction)
    await portfolioRepo.save(portfolio)
    await userRepo.save(user)

    console.log("User has been saved");
  })
}

```

Code pour l'inscription d'un usager

Un des développements à venir est l'implémentation des transactions. L'utilisateur devra être en mesure de simuler des achats ou des ventes de crypto-monnaies.

Le défi le plus significatif sera de gérer les données reçues après avoir recherché une monnaie afin de faire les calculs nécessaires pour les transactions.

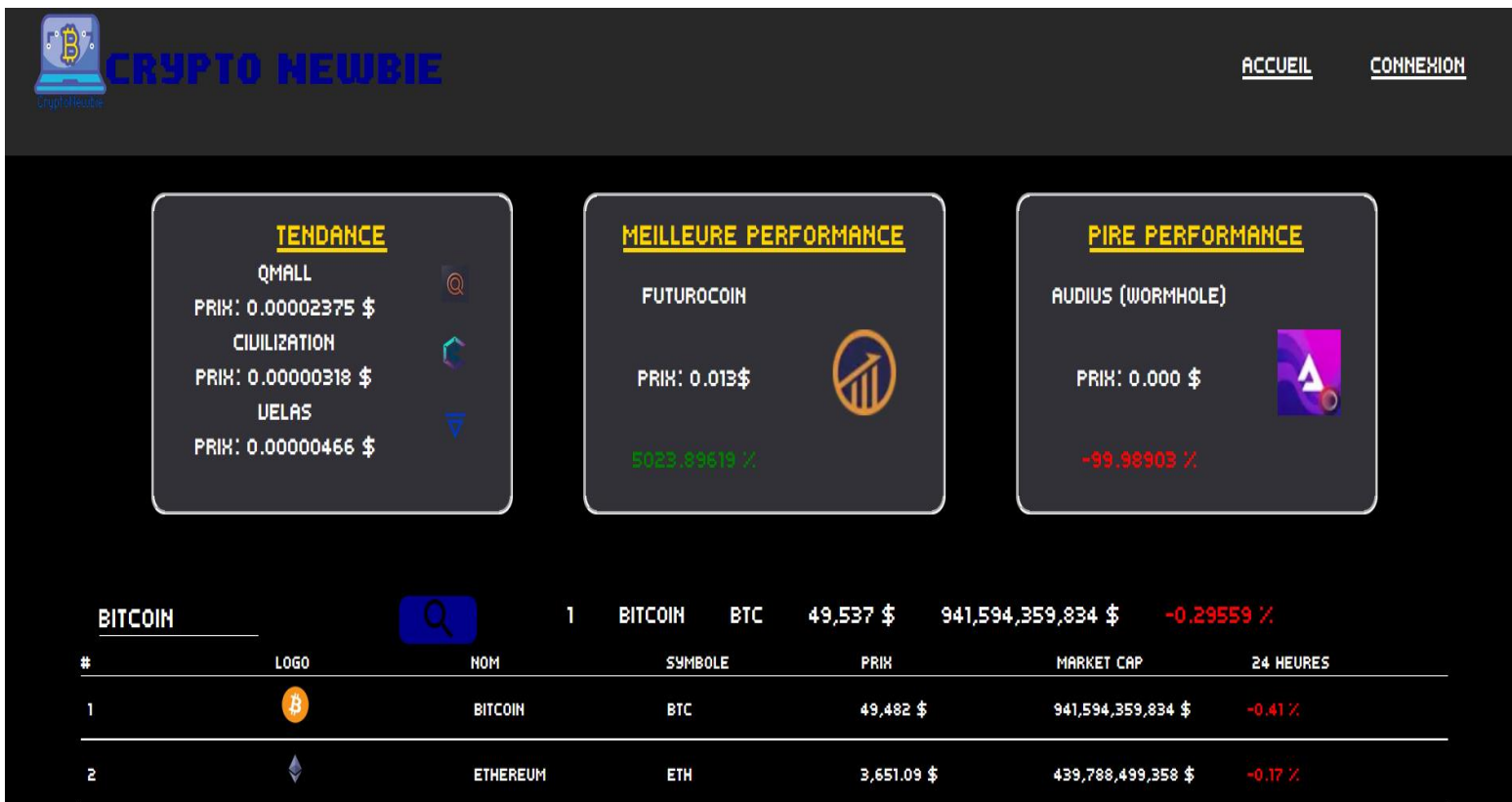
## Nikolas

Une des fonctionnalités implémentées par Nikolas est une fonction permettant de rechercher une crypto-monnaie, après avoir implémenter les premiers appels à l'API de CoinGecko.

Le plus grand défi a été d'intégrer la fonction dans un composant React. Cette bibliothèque étant nouvelle, l'apprentissage fut quand même difficile au début. Quel type de composant utiliser, comment faire interagir deux composants ensemble ou comment utiliser les Hook de React sont le genre de murs auxquels on s'est souvent frappé.

Le code pour cette fonctionnalité se trouve sous :

- Le code du composant de recherche : `/frontend/components/searchBar.tsx`
- Le code de l'implémentation du composant : `/pages/index.tsx`



Exemple d'une recherche

```

export default function SearchBar() {
  const [handlerState, setHandler] = useState({market_data:{current_price:{cad: ""}, market_cap_rank:"", market_cap:{cad: ""}, price_change_percentage_24h: ""}});
  const [search, setSearch] = useState("");

  const updateSearch = (e) => {
    setSearch(e.target.value);
  };

  const coingeckoUrl = (coin) => {
    coin = String(coin);
    return `https://api.coingecko.com/api/v3/coins/${coin.toLowerCase()}/`;
  };

  const coingeckoFetch = () => {
    if (search !== ""){
      fetch(coingeckoUrl(search)).then((response) => {
        response.json().then((jsonData) => {
          if (jsonData.error !== "Could not find coin with the given id"){
            setHandler(jsonData);
            let divResult = document.querySelector(".search-result");
            divResult.style.visibility = "visible";
          }
          else{
            alert("La recherche n'est pas valide !");
          }
        })
      });
    }
    else {
      alert("Veuillez spécifier une crypto");
    }
  };

  return (
    <div>
      <div className='search-section'>
        <div className='input-search'>
          <input type="text" id="search" name="search" onChange={updateSearch} placeholder="Rechercher" />
        </div>
        <div className='button-search'>
          <button onClick={coingeckoFetch}><Image src={"/search-icon.png"} width={"32px"} height={"32px"}/></button>
        </div>
      </div>
      <div className='search-result'>
        <li title="Le rang">{handlerState.market_data.market_cap_rank}</li>
        <li title="Le nom">{handlerState.name}</li>
        <li title="Le symbol">{handlerState.symbol}</li>
        <li title="Le prix en cad">{handlerState.market_data.current_price.cad.toLocaleString() + ' $'}</li>
        <li title="La capitalisation">{handlerState.market_data.market_cap.cad.toLocaleString() + ' $'}</li>
        <li title="L'évolution en 24h en %" style={{ color: Math.sign(handlerState.market_data.price_change_percentage_24h) === -1 ? 'red' : 'green' }}>{handlerState.ma
      </div>
    </div>
  )
}

```

Composant de recherche

Un des développements à venir est l'implémentation du portfolio des inscrits. L'utilisateur devra être en mesure de suivre l'évolution de ses avoirs.

Le défi le plus significatif sera de générer les différents graphs qui donneront un support visuel de l'évolution du portfolio de l'utilisateur.

Somme toutes, nous estimons avoir complété environ 60% du projet, conception et planification incluses.