EPITECH INNOVATIVE PROJECT

DOCUMENTATION TECHNIQUE

PROMO 2024

Description du document

Titre	Documentation Technique
Objet	Instruction pour l'utilisateur
Auteur	Block2SChool
E-mail	block2school_2024@labeip.epitech.eu
Mots-clés	Documentation, utilisateur
Promotion	2024
Date de modification	29 juillet 2023
Version du document	1.0

Table des révisions

Date	Version	Auteur	Section(s)	Commentaires
29/07/2023	1.0	Block2School	All	First Version

Table des matières

1. Présentation du projet

1.1. EIP

L'EIP (Epitech Innovative Project) est le projet de fin d'études des étudiants d'EPITECH de 2 ans.
Il débute en 3ème année et se termine au milieu de la 5ème année. Il se déroule en groupe de 6 personnes minimums, chaque groupe choisit son sujet et doit le développer de manière autonome et professionnelle.
L'objectif de l'EIP est d'apprendre aux élèves d'EPITECH la gestion d'un projet dans son intégralité. De cette manière, tous les élèves prennent connaissance des différentes étapes de conception, réflexion et communication d'un projet de grande envergure : l'EIP permet aux étudiants d'acquérir une expérience intégrale de la gestion de projets par la pratique.

1.2. Notre projet

Block2School est un projet qui vise à démocratiser le développement, ainsi que la blockchain.
Il permet via un site web, de s'initier à la programmation dans divers langages comme le Javascript, C, Python ou encore Solidity (langage utilisé pour développer des 'smarts-contract' sur plusieurs blockchains comme Ethereum ou Binance Smart Chain).
Notre site offre également un Marketplace dans lequel il est possible d'acheter des NFT créés spécialement pour notre plateforme, mais aussi de les échanger. Sur le Marketplace, vous pouvez également acheter de la cryptomonnaie Block2School (B2ST) pour soutenir le projet et dans le futur acheter des goodies directement sur le site.
Nous avons également une partie blog qui sert à échanger avec les utilisateurs afin de pouvoir répondre à leurs questions, leur partager les nouveautés de l'application ainsi que des informations concernant l'univers de la blockchain.

2. Prérequis

Pour pouvoir développer sur le projet, vous aurez besoin de :

2.1.Liens importants

Voici quelques liens dont vous aurez besoin :

- http://eip.epitech.eu/2024/block2school → Site vitrine
- http://51.77.194.105:3000 → Site web

3. Stack technique

- Git / Github

3.1. Front

La stack technique du Front est composée de :
- Typescript
- Nextjs

Nous utilisons plusieurs bibliothèques / paquets → chakra-ui / ethers …

3.2. Backend

La stack technique du Backend est composée de :
- Python
- FastAPI
- MySQL

3.3. Code Executer

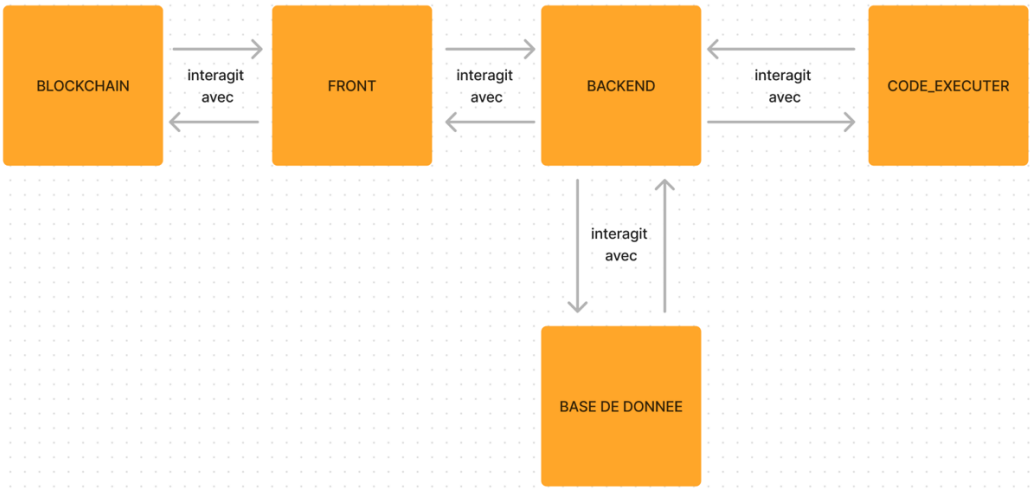
La stack technique du Code executer est composée de :
- Docker
- Javascript
- Express.js

3.4. La blockchain

Pour le côté Blockchain, nous utilisons la Blockchain de Binance (Binance Smart Chain), qui utilise le langage Solidity pour l'écriture des contrats NFTs et Token. Nous utilisons donc ce langage de programmation pour créer nos contrats.

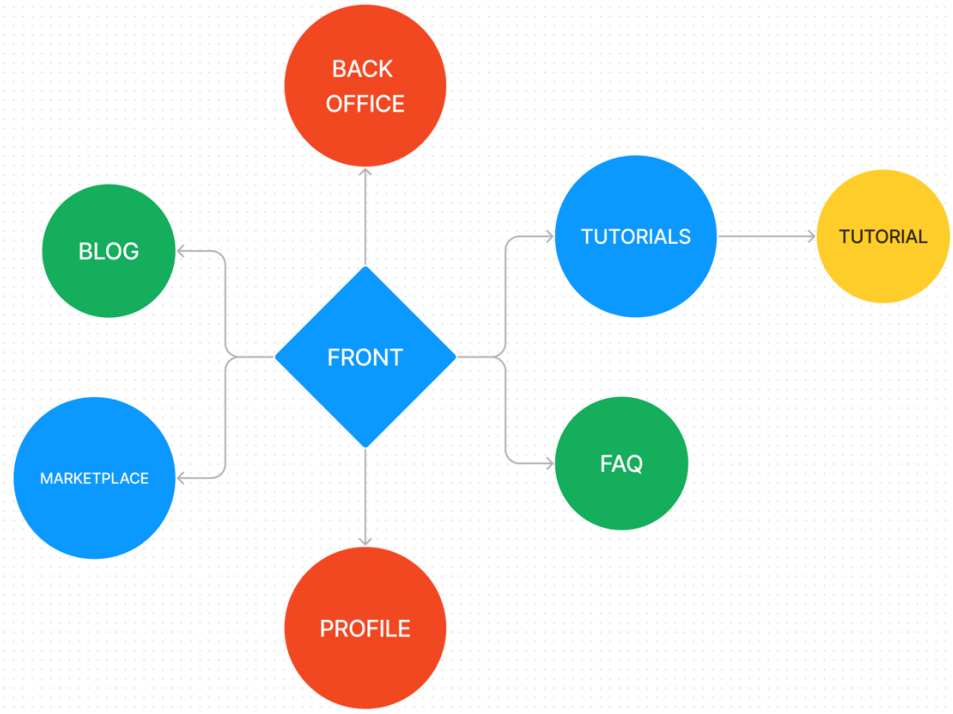
4. Architecture

4.1. Architecture globale

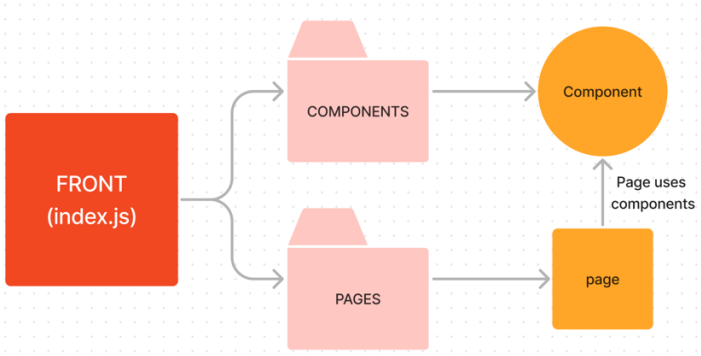


4.2. Site web

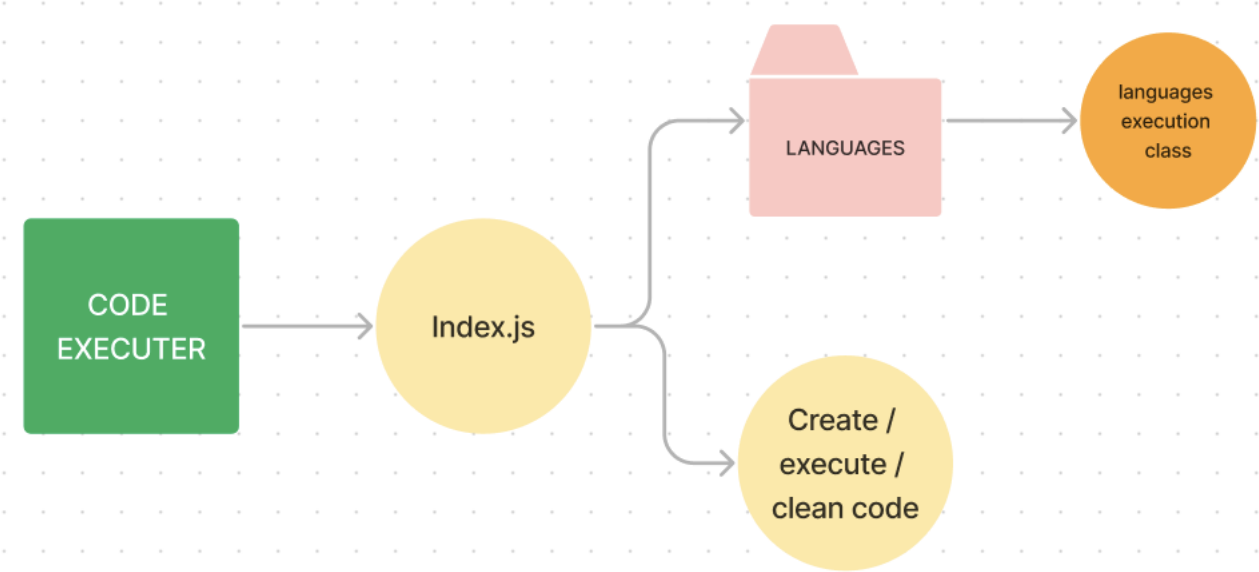
4.2.1. Composition du site



4.2.2. Organisation du Front



4.2.3. Code executeur



5. Contribuer

Pour contribuer au projet, il vous faudra suivre les instructions suivantes :

5.1. Outils à utiliser

Pour gérer les versions, les repositories ect, nous utilisons git et github.

5.2. Bonnes pratiques

Lorsque vous travaillez sur une des parties du projet, vous devez suivre les règles suivantes :

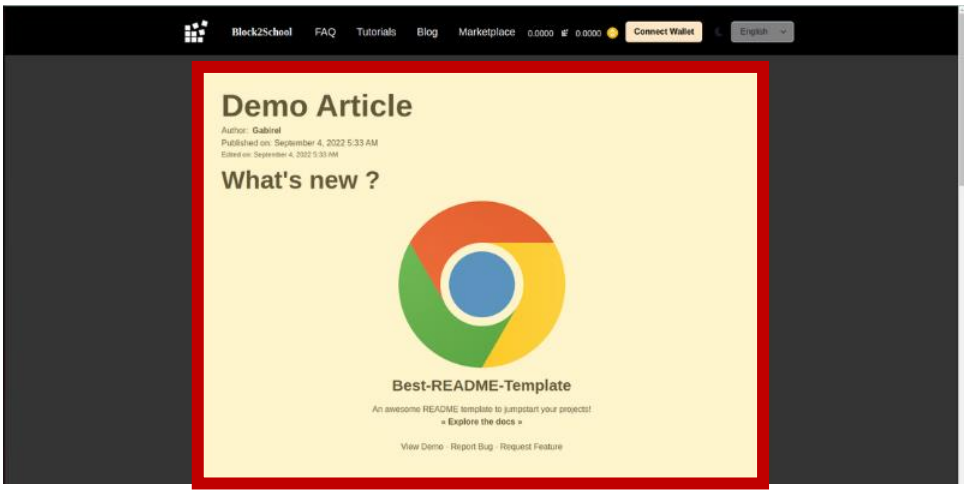
1. Créez une nouvelle branche
2. Une fois votre travail effectué, testez bien vos modifications
3. Créez une pull request sur le repo github
4. Assignez une personne à cette pull request.

6. Mise en production

7. Front-End Components

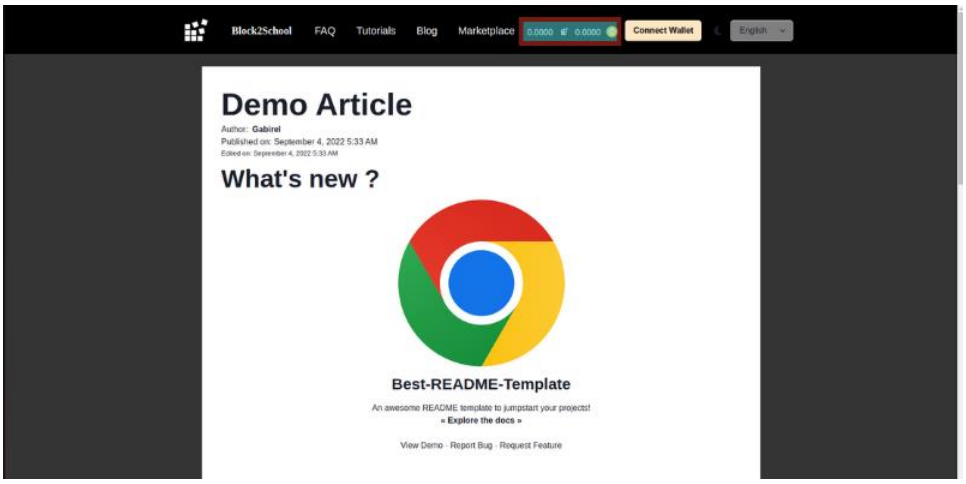
Utilisation de chakra-ui pour tout le Front-End voir : https://chakra-ui.com/docs/components

7.1.ArticleBody



ArticleBody – Arguments :
articleTitle: any, articleAuthor: any, articlePublicationDate: any, articleEditDate: any, markdownSource: string

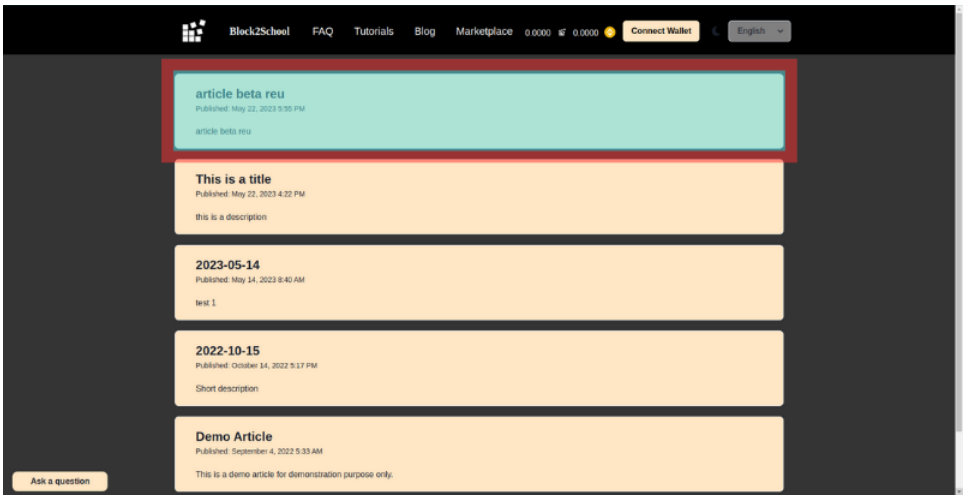
7.2.NavbarBalance & NavbarAllBalances



NavBarBalance – Arguments : balanceToken:any, ImgSrc:any, alt:any, _decimal: number

NavBarAllBalances – Arguments : NONE

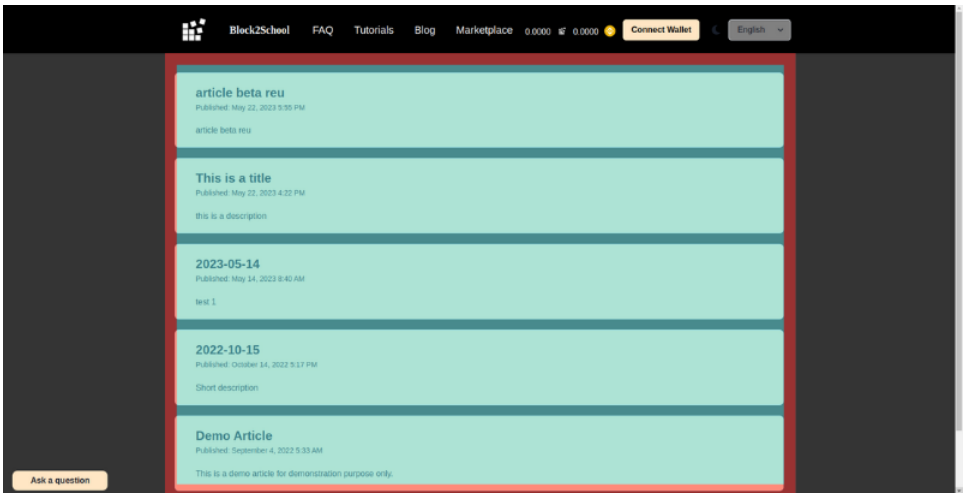
7.3.BlogCard



BlogCard – Arguments :

id: number, title: string, author: string, markdownUrl: string, publicationDate: number, shortDescription: string, editDate: number

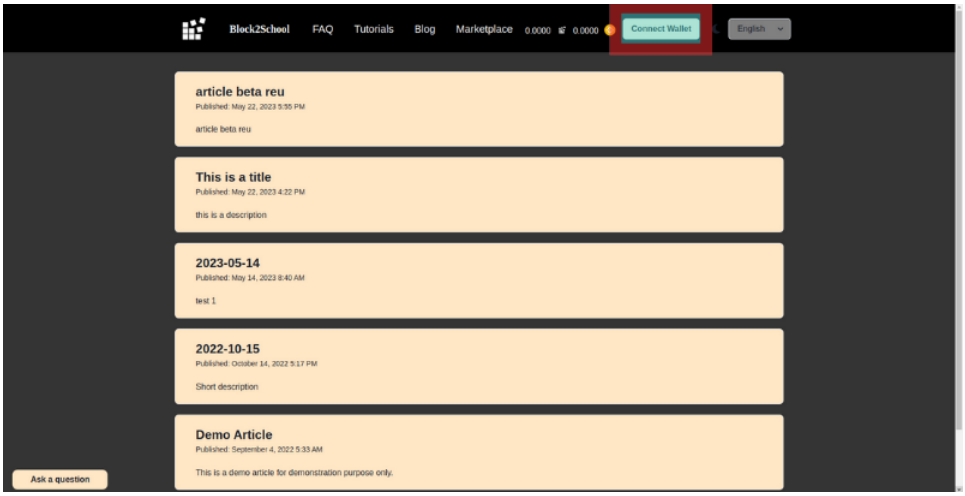
7.4.BlogList



BlogList – Arguments:

articles:any

7.5.CustomButton & ConnectionButton



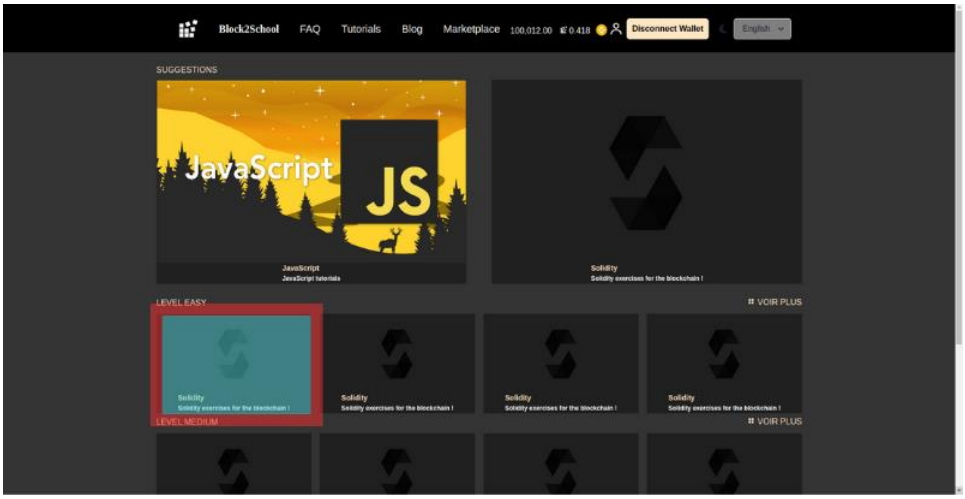
CustomButton – Arguments:

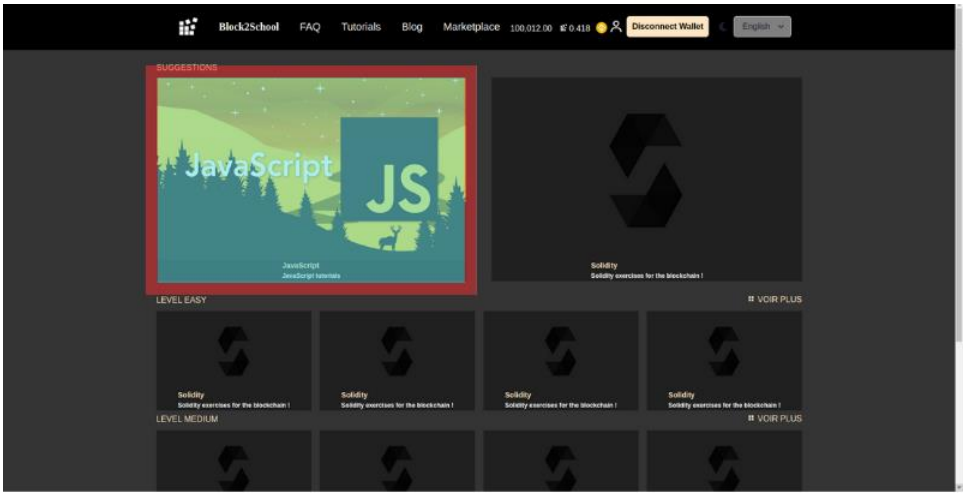
gap:any, id:any, name:any, srcImg:any, alt:any ,size:any, disabled:any, variant:any, onClick:any, hImg:any, wImg:any,borderRadius:any

In this case the button is the ConnectionButton

Arguments – None

7.6.TutorialCategoryCard & TutorialCategoryCardSmall

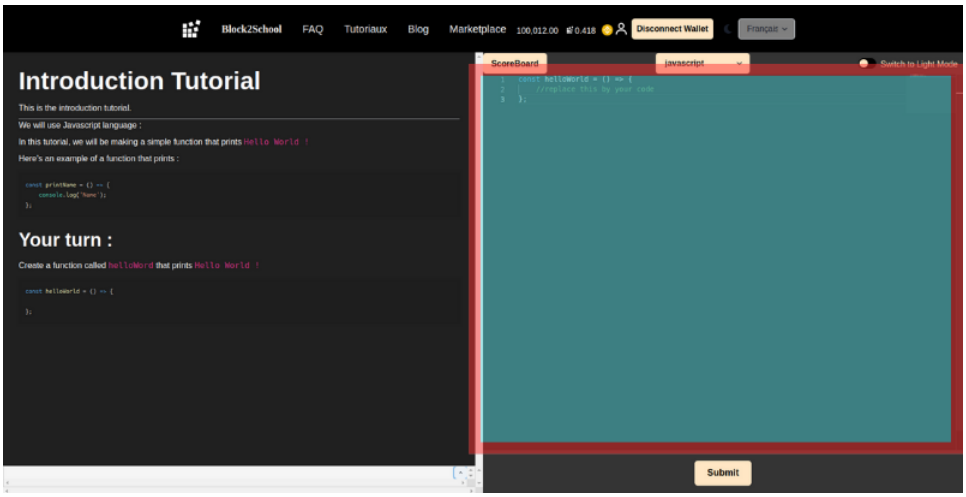




TutorialCategoryCard & TutorialCategoryCardSmall – Arguments :
name: string, image: string, description: string, callback: any

TutorialCategoryCardatIndex & TutorialCategoryCardSmallatIndex – Arguments:
categories:any, index:number, showCategoryTutorialsModal:any

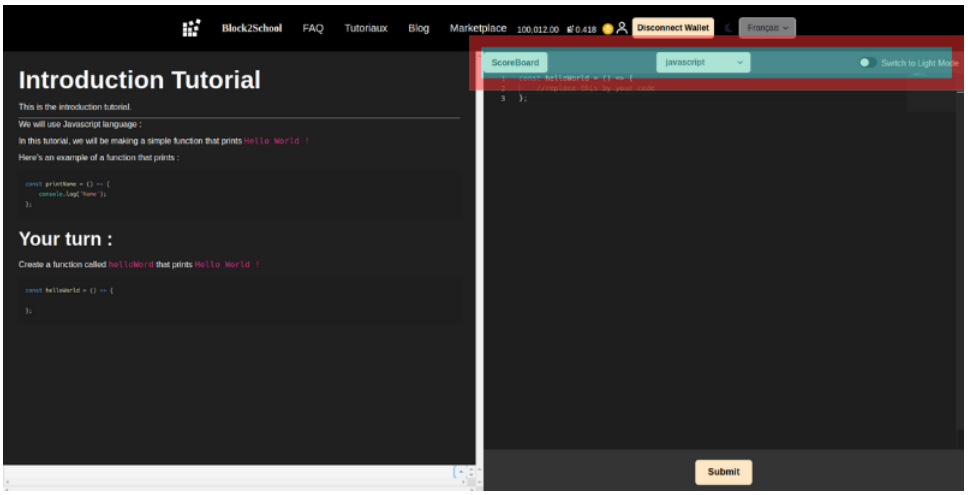
7.7.Editor



Editor – Arguments :

theme: any, lang: any, onChange: any, defaultValue: any, onMount:any, options:any

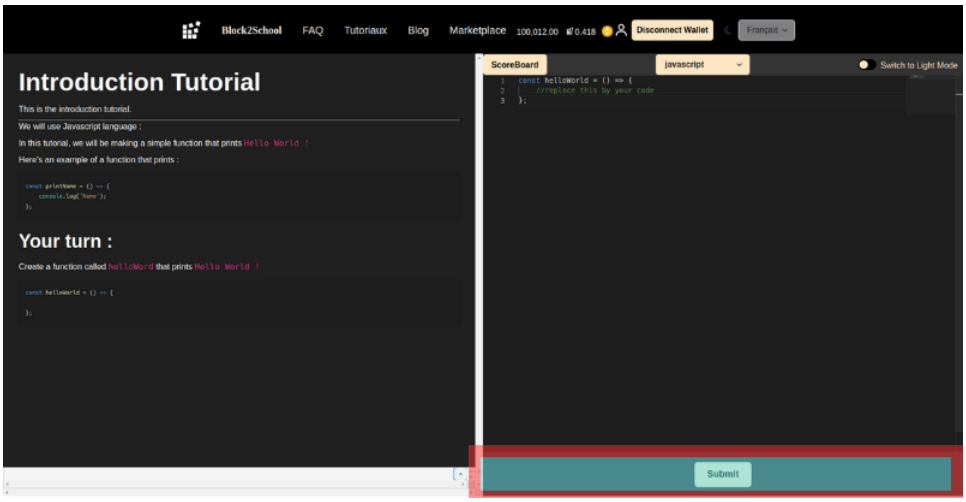
7.8.OptionEditor



OptionEditor – Arguments :

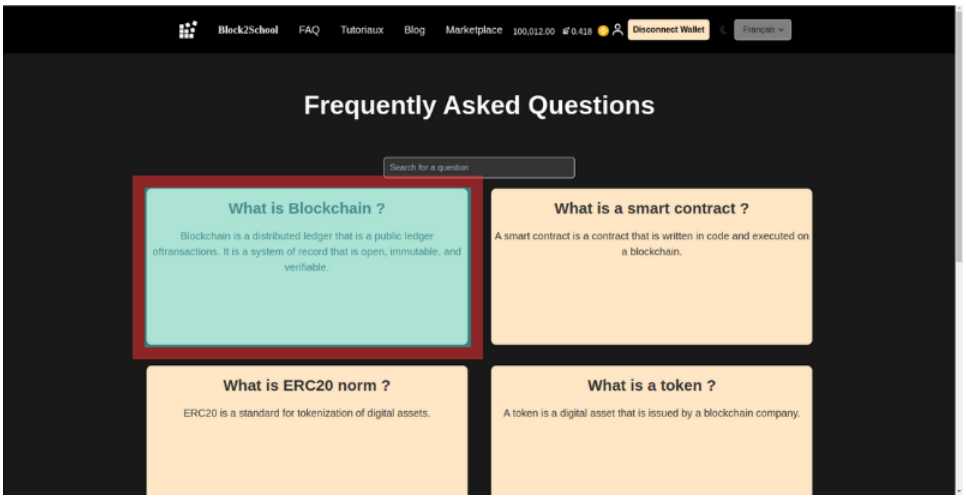
changeLang:any, scoring:any, switchText:any, changeTheme:any

7.9.UploadEditor



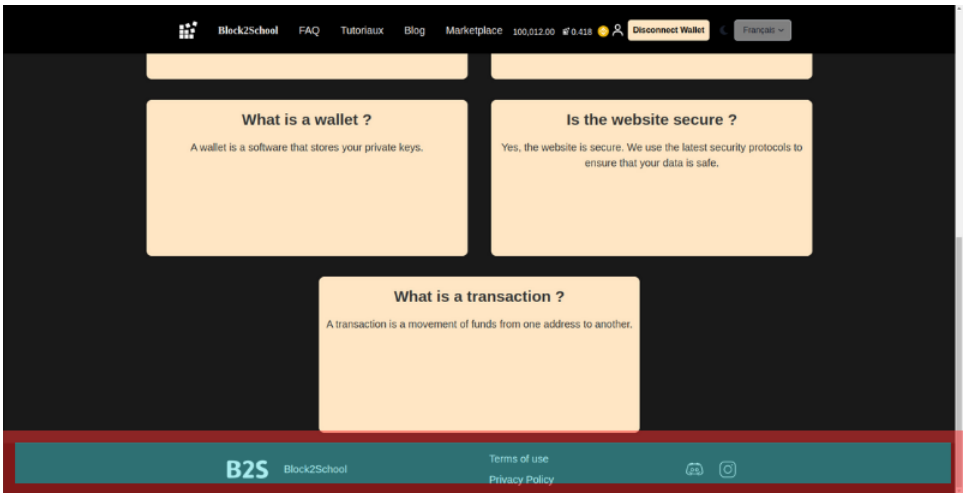
UploadEditor – Arguments :
isUploading:any, uploadCode:any

7.10. Question



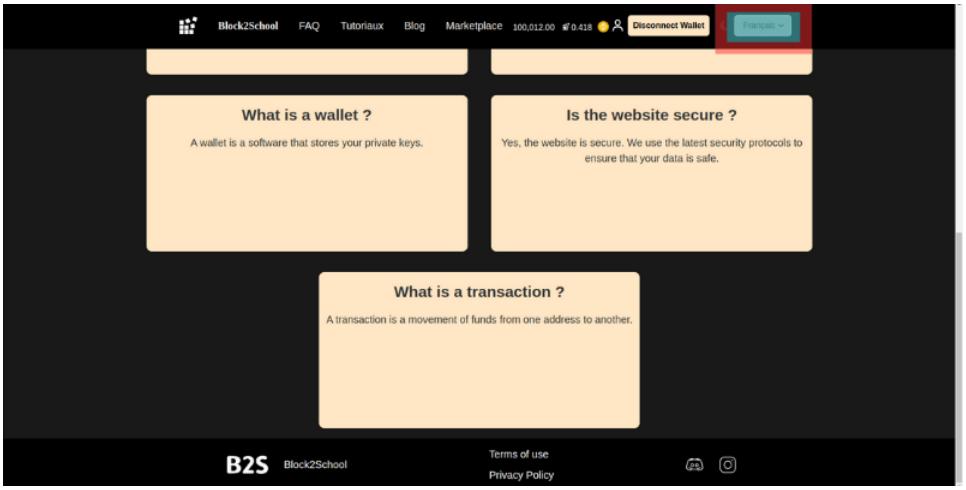
Question – Arguments :
question:any, answer:any

7.11. Footer



Footer – Arguments : NONE

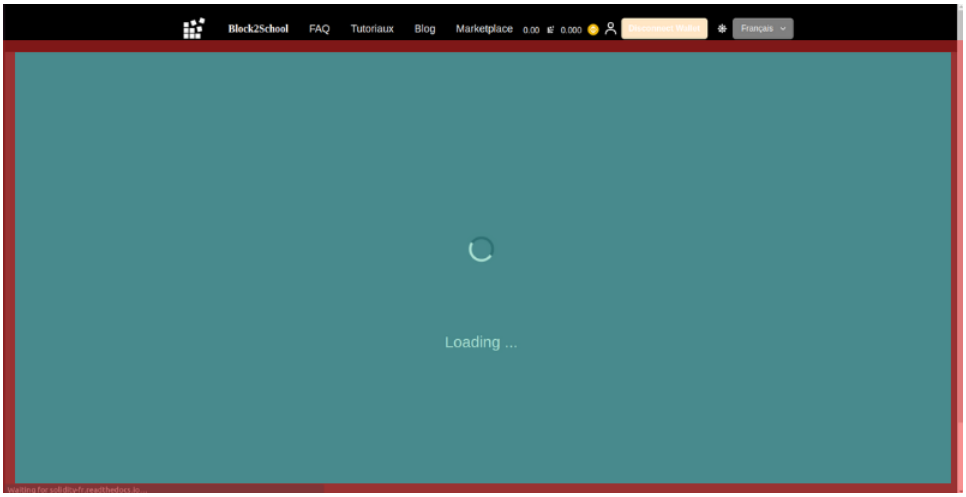
7.12. LanguageSwitcher & LanguageProvider



LanguageSwitch – Argument : None

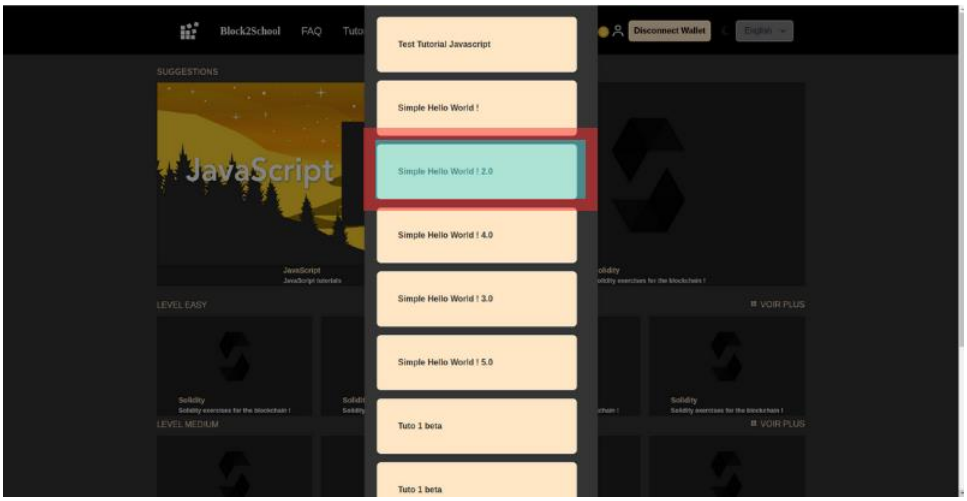
LanguageProvider – Argument :
children : LanguageProviderProps

7.13. LoadingScreen



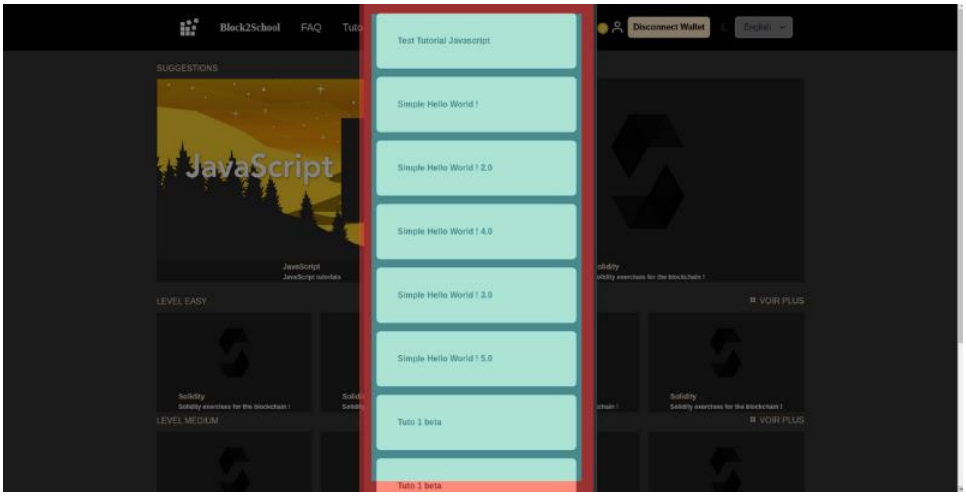
LoadingScreen – Arguments :
showError:boolean

7.14. TutorialCards



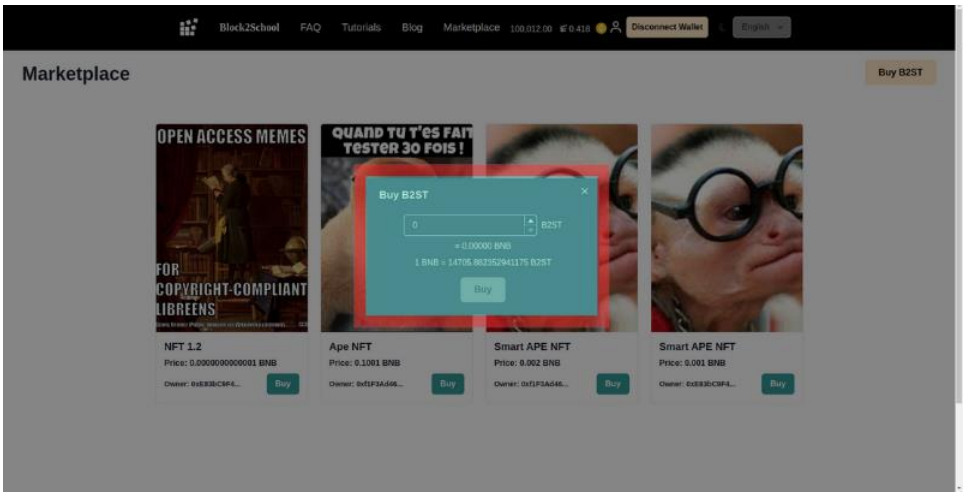
TutorialCards – Arguments:
tutorials:any

7.15. TutorialCategoryModal



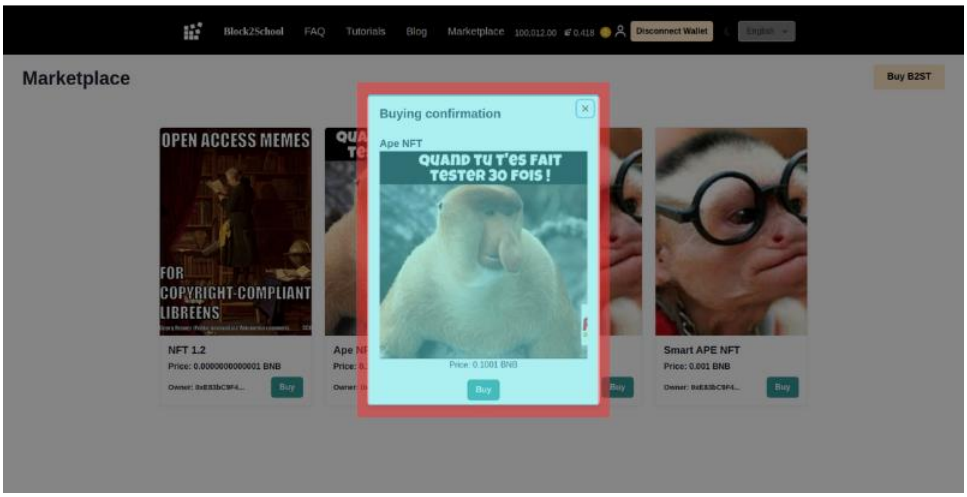
TutorialCategoryModal – Arguments:
isOpen: boolean, closeModal: any, category: string

7.16. BuyB2STokenModal



BuyB2sTokenModal – Arguments:
isOpenModal: boolean, closeModal: any

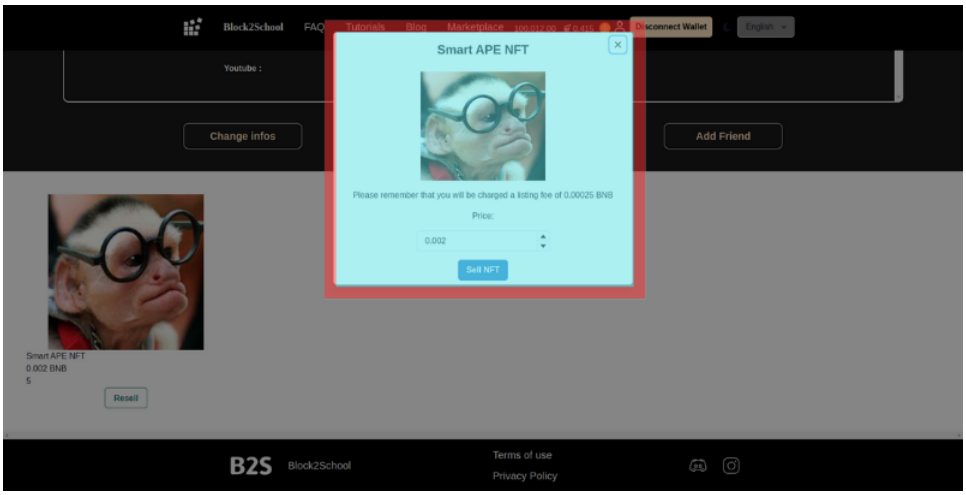
7.17. BuyNFTModal



BuyNFTModal – Arguments:

isOpenModal: boolean, closeModal: any, _nft: NFT

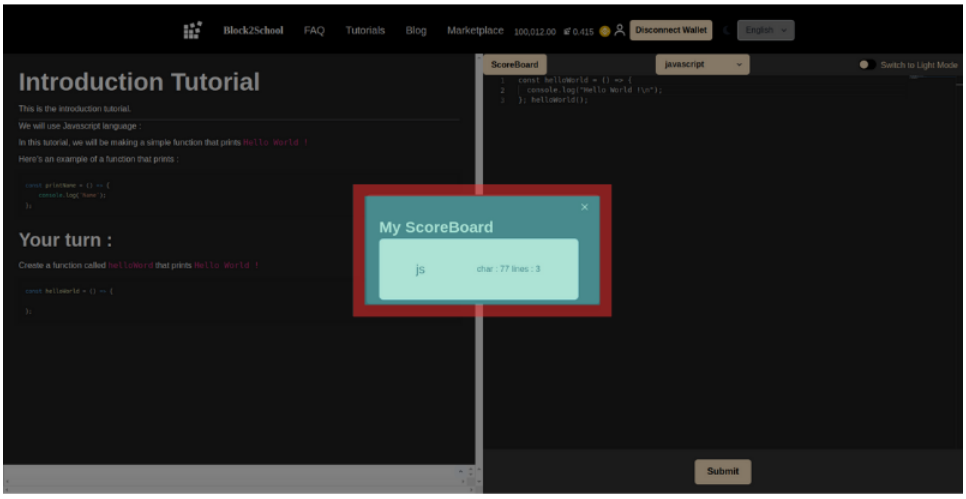
7.18. MyNFTModal



MyNFTModal – Arguments:

isOpenModal: boolean, closeModal: any, _nft: NFT, listingPrice: string

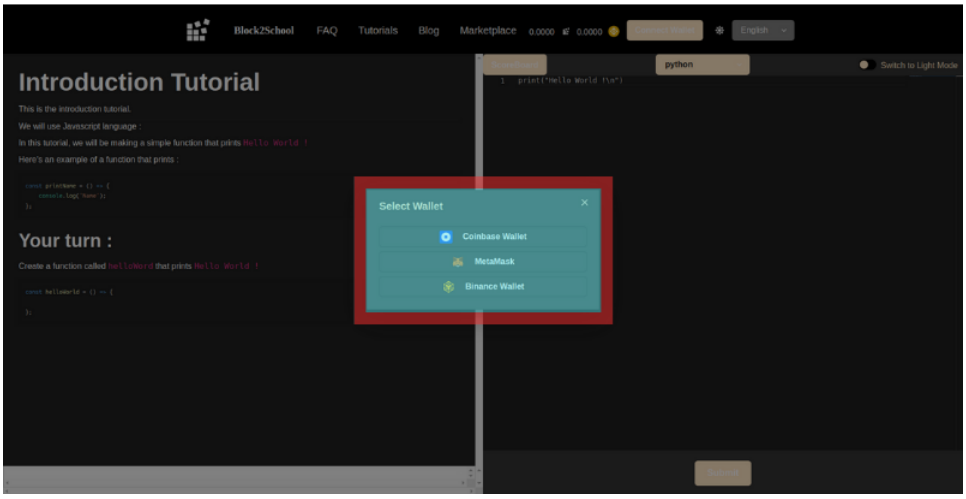
7.19. ListScores & ScoreBoardModal



ListScores – Arguments :
score:any

ScoreBoardModal – Arguments:
isOpen:boolean, closeModal:any, scoreboard:any

7.20. LoginOptions & WalletModal

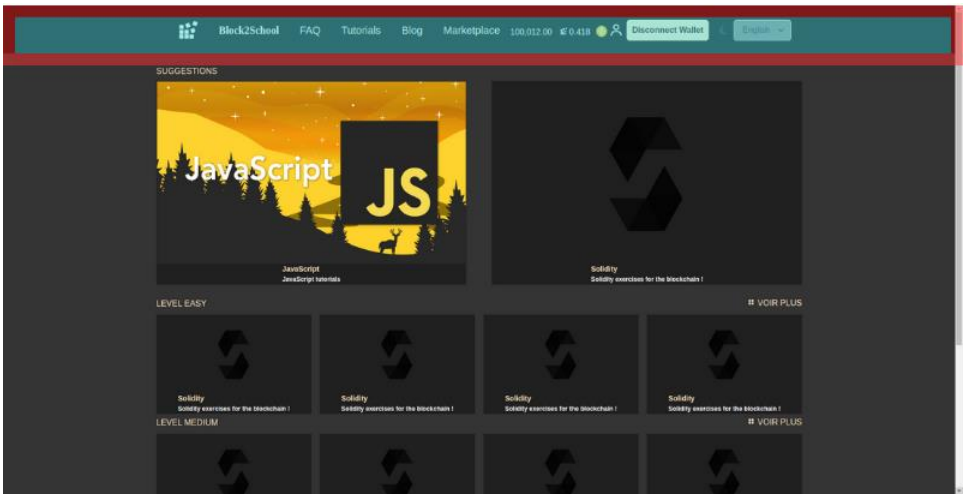


LoginOptions – Arguments :
isOpen: boolean, closeModal: any

WalletModal – Arguments :

isOpen: boolean, closeModal: any

7.21. NavBar

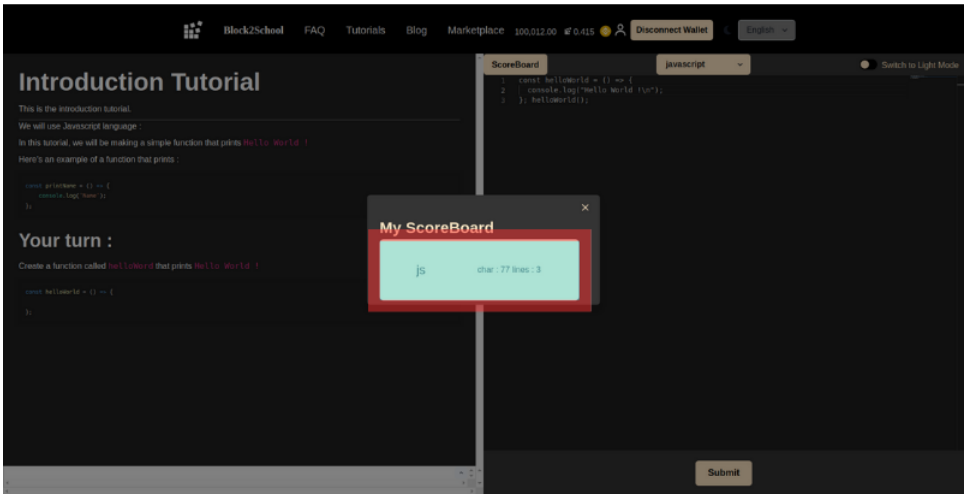


NavBar – Arguments : NONE

7.22. UserNFTView

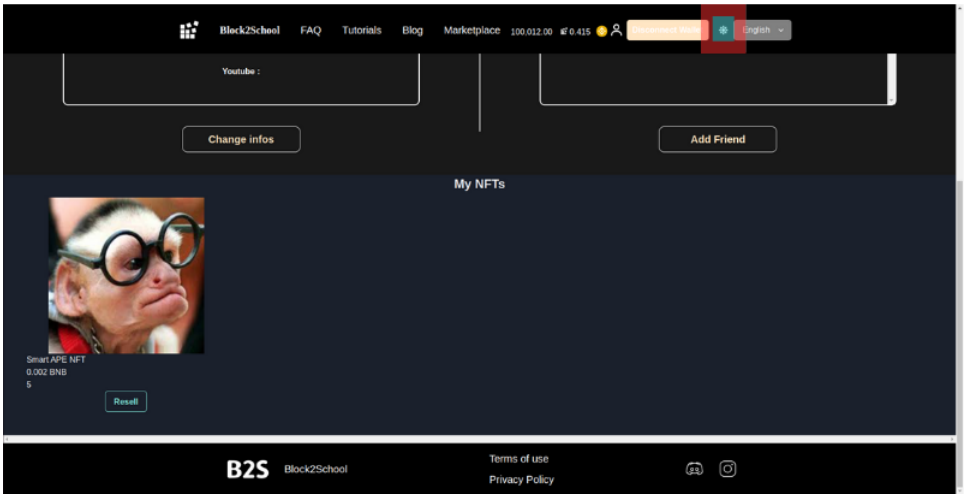
UserNFTView – Arguments: NONE

7.23. ScoreCard



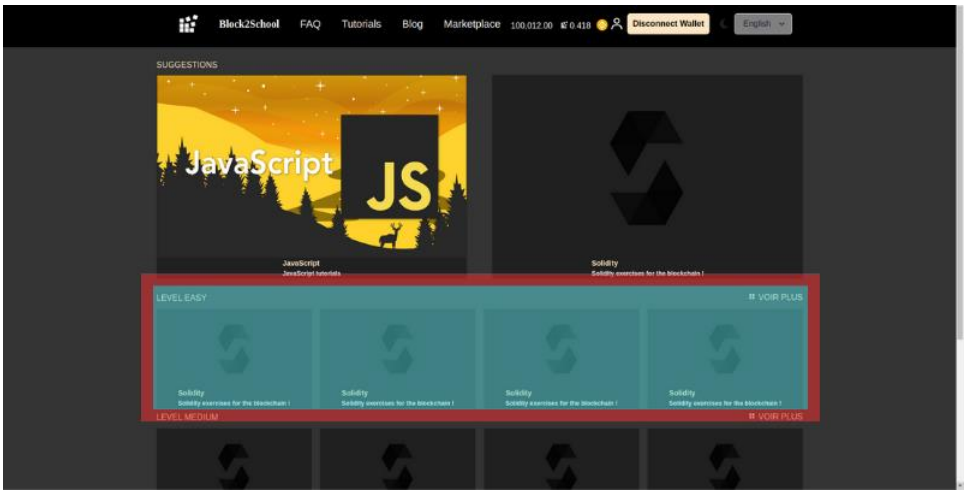
ScoreBoard – Arguments :
language:string, characters:number, lines:number

7.24. ThemeSelector



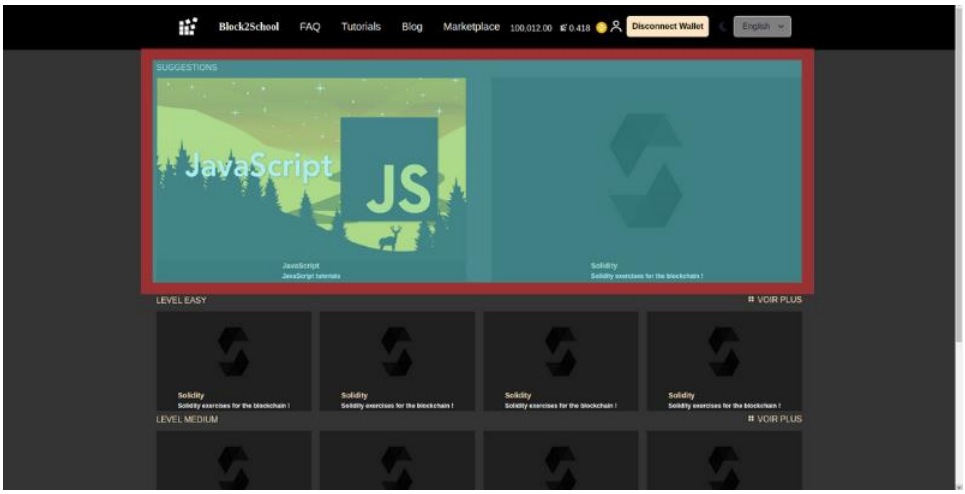
ThemeSelector – Arguments :
props: ThemeSelectorProps

7.25. LevelTutorial



LevelTutorial – Arguments :
name:any, categories:any, showCategoryTutorialsModal:any

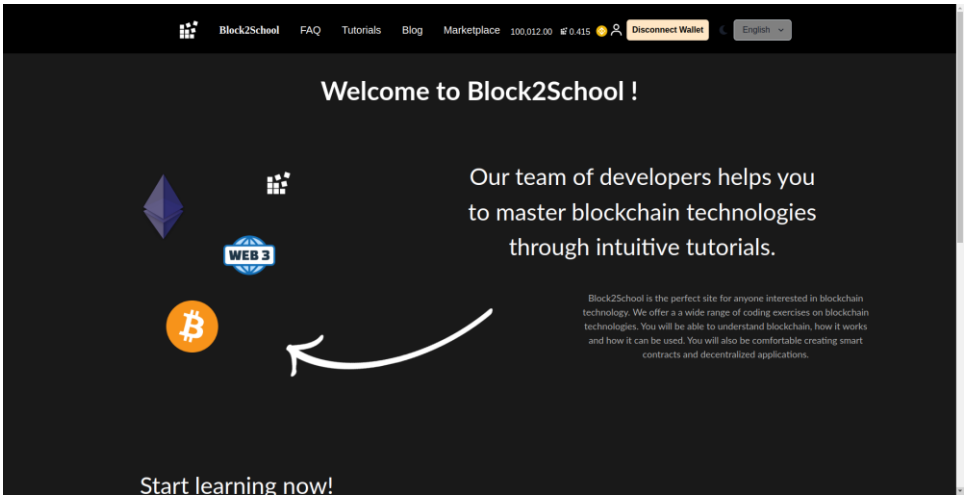
7.26. SuggestionTutorial



SuggestionTutorial – Arguments :
categories:any, showCategoryTutorialsModal:any

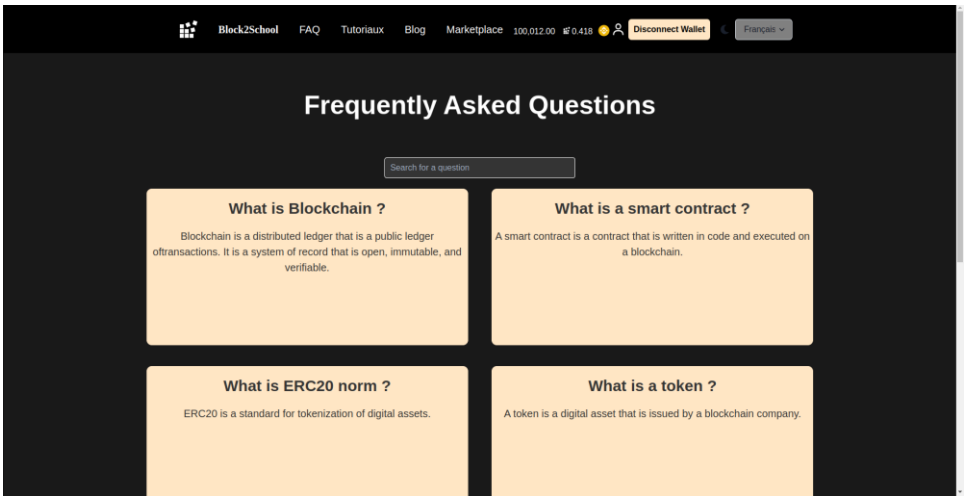
8. Front End – Pages

8.1. Home



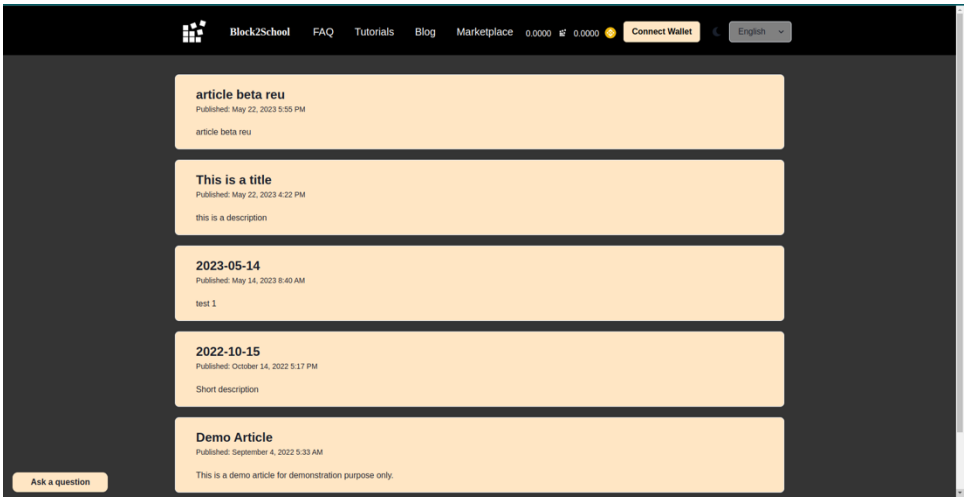
app/pages/home.tsx

8.2. FAQ



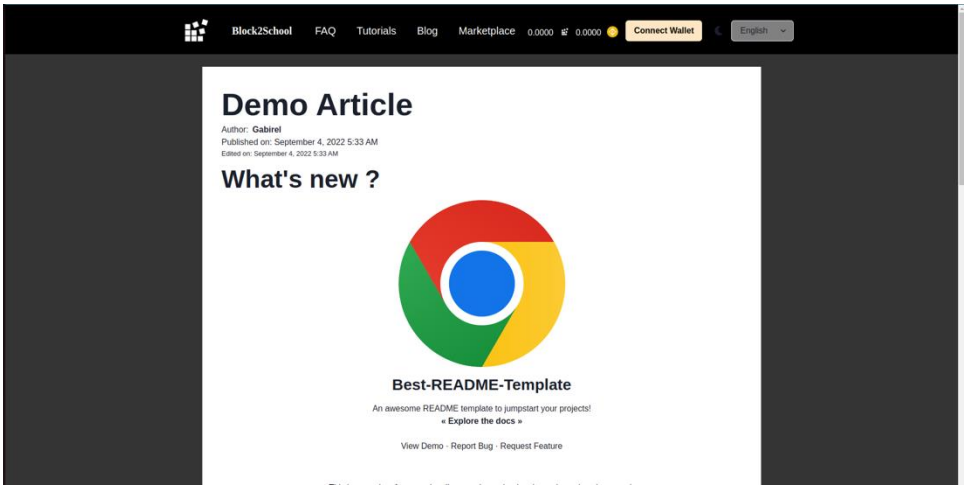
app/pages/faq.tsx

8.3.Blog



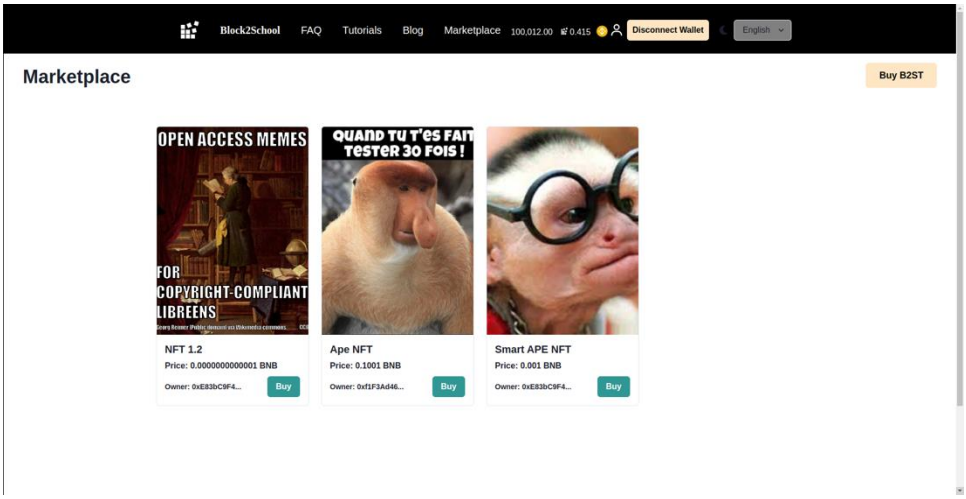
app/pages/blog.tsx

8.4.Article



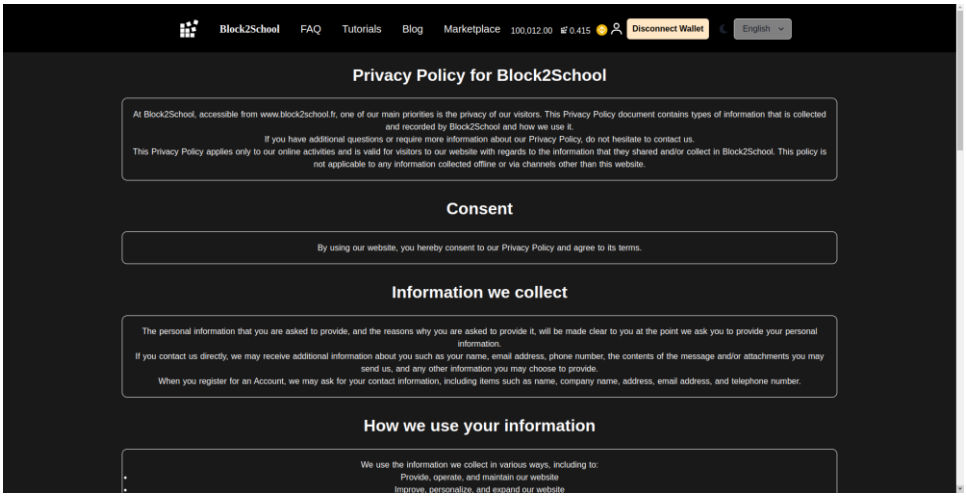
app/pages/article.tsx

8.5.MarketPlace



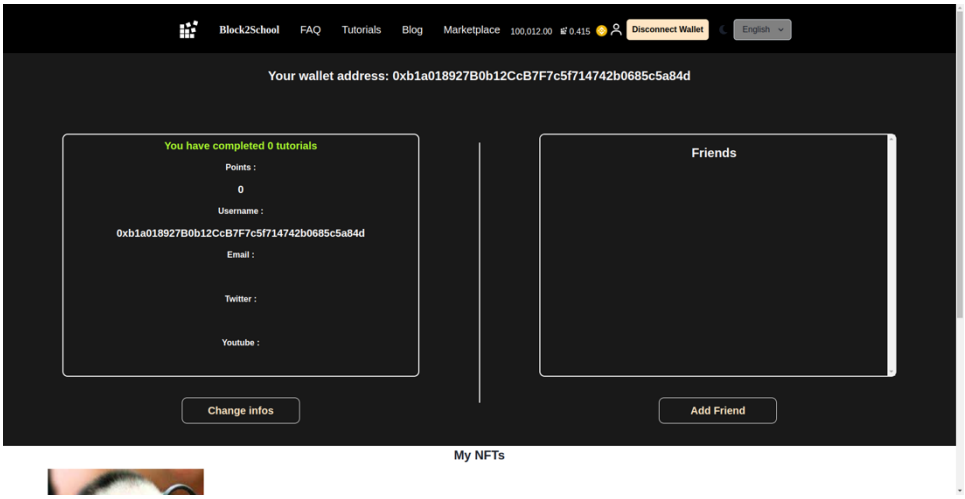
App/pages/marketplace.tsx

8.6.Privacy & Policy



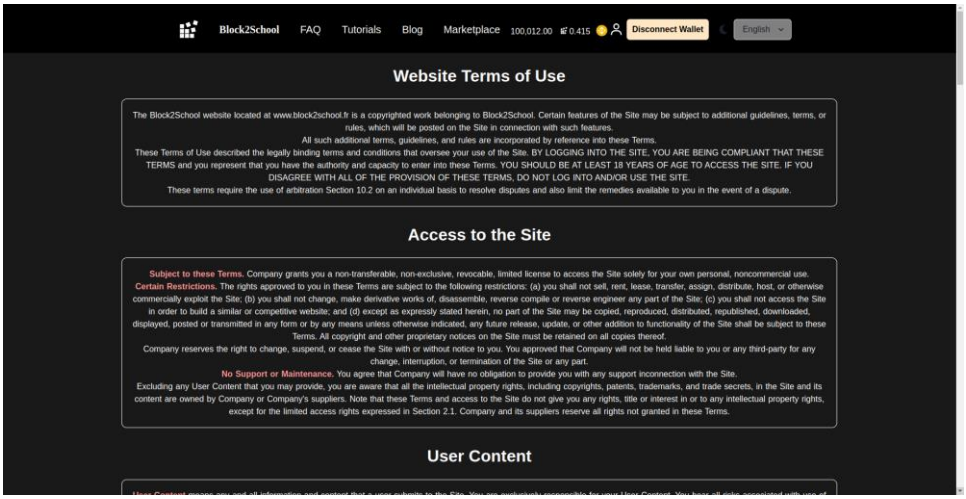
App/pages/privacy-policy.tsx

8.7.Profile



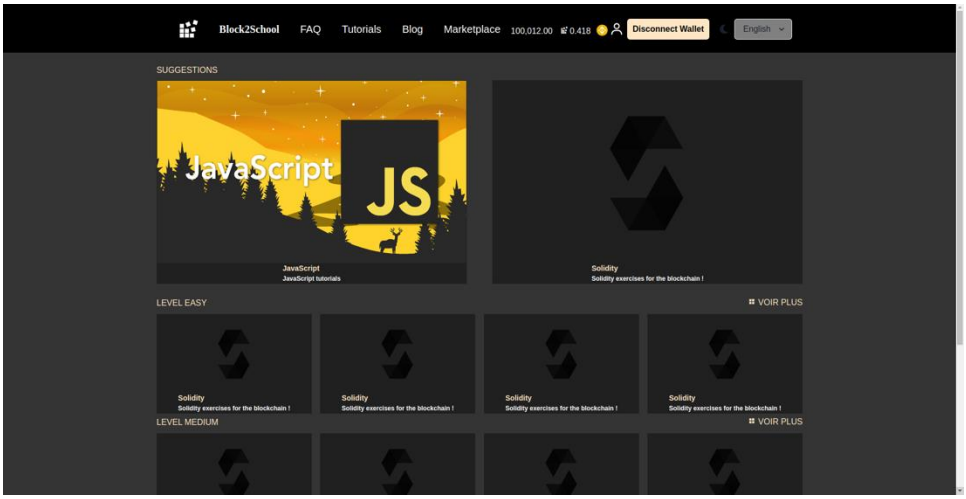
App/pages/profile.tsx

8.8.Terms of Use



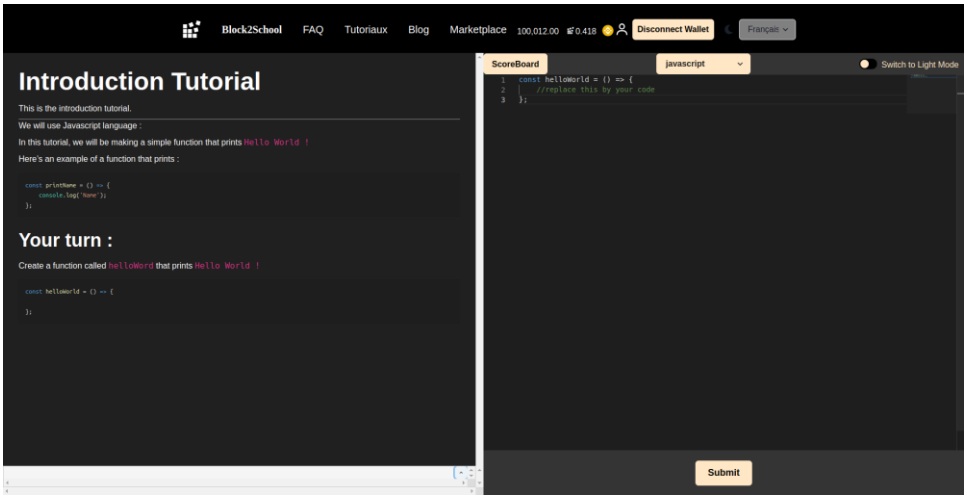
App/pages/terms-of-use.tsx

8.9.Tutorials



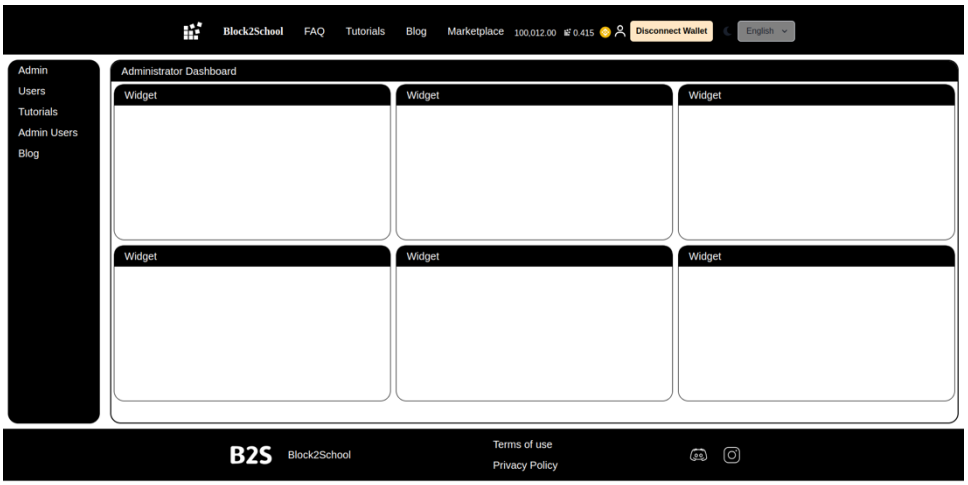
App/pages/tutorials.tsx

8.10. Tutorial



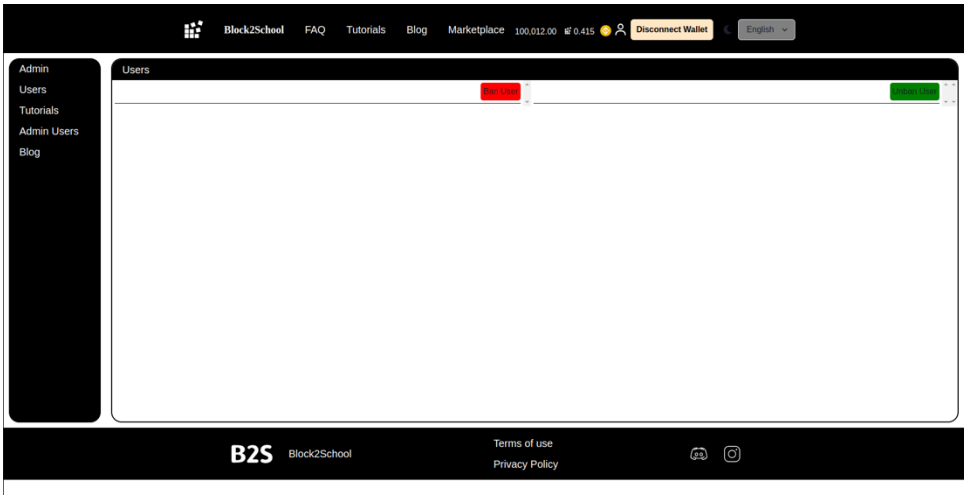
App/pages/tutorial.tsx

8.11. Back Office



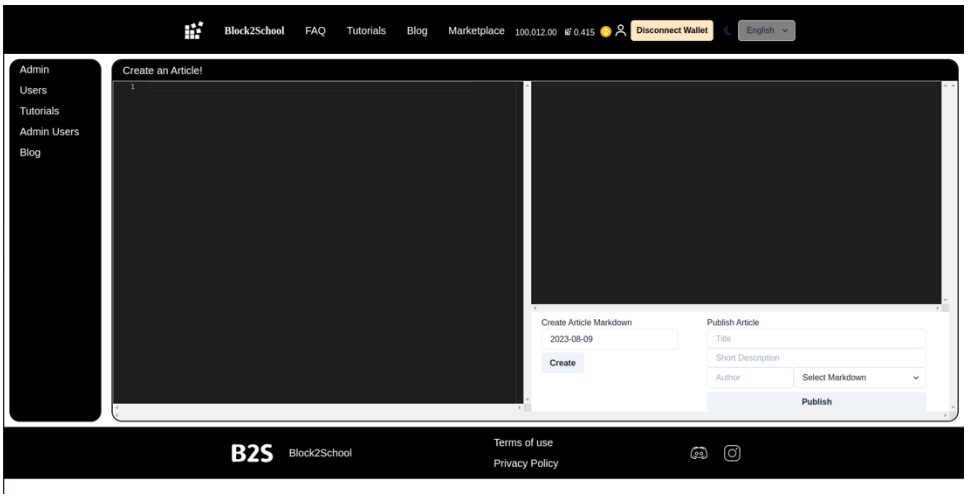
App/pages/back-office.tsx

8.12. Users



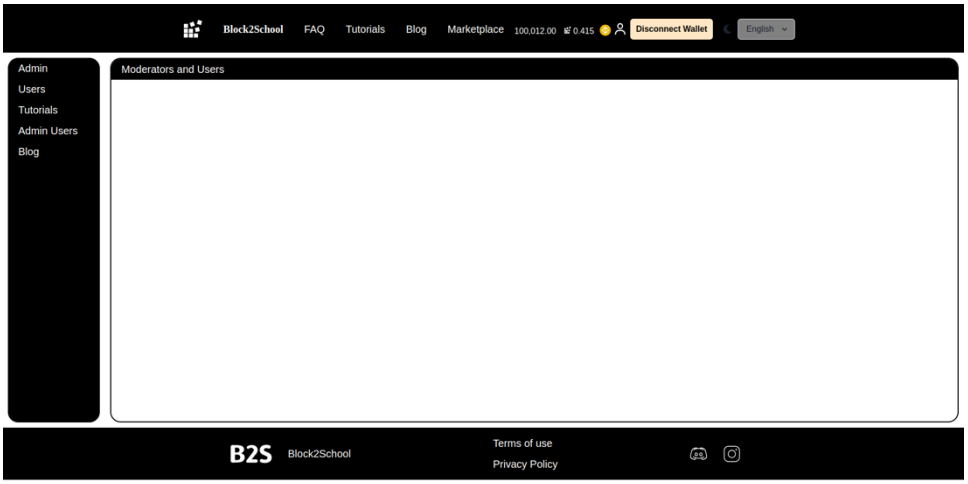
App/pages/back-office/users.tsx

8.13. Admin Blog



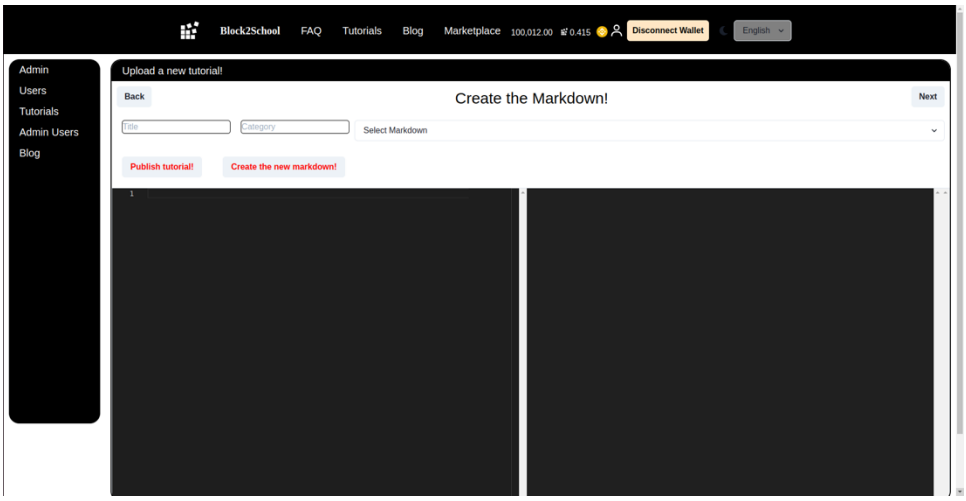
App/pages/back-office/blog.tsx

8.14. Admin Users



App/pages/back-office/admin-users.tsx

8.15. Admin Tutorials



App/pages/back-office/tutorials.tsx

9. Backend autorisations

9.1.JWTChecker

La classe JWTChecker est utilisée pour vérifier le jeton d'authentification des routes utilisateurs.
Le jeton contient l'UUID et la date d'expiration du jeton comme ceci:
{"uuid": string, "expires": number}
La validité du jeton est de 2 heures.
Lors de la vérification du jeton, si ce dernier est invalide, une erreur 401 sera retournée pour toute route nécessitant l'authentification JWTChecker.

9.2.AdminChecker

La classe AdminChecker est utilisée pour vérifier le jeton d'authentification des routes administrateurs.
Le jeton contient également l'UUID et la date d'expiration au format suivant:
{"uuid": string, "expires": number}
La validité du jeton est de 2 heures.
En plus de la vérification du JWTChecker, une requête vers la base de données est effectuée avec l'UUID contenu dans le jeton afin de vérifier si l'utilisateur est bien administrateur.
Il y a deux niveaux d'authentification: Administrateur et Modérateur, défini lors de la création de la route.
1 = Modérateur
2 = Administrateur

10. Hashing du wallet et encodage du JWT

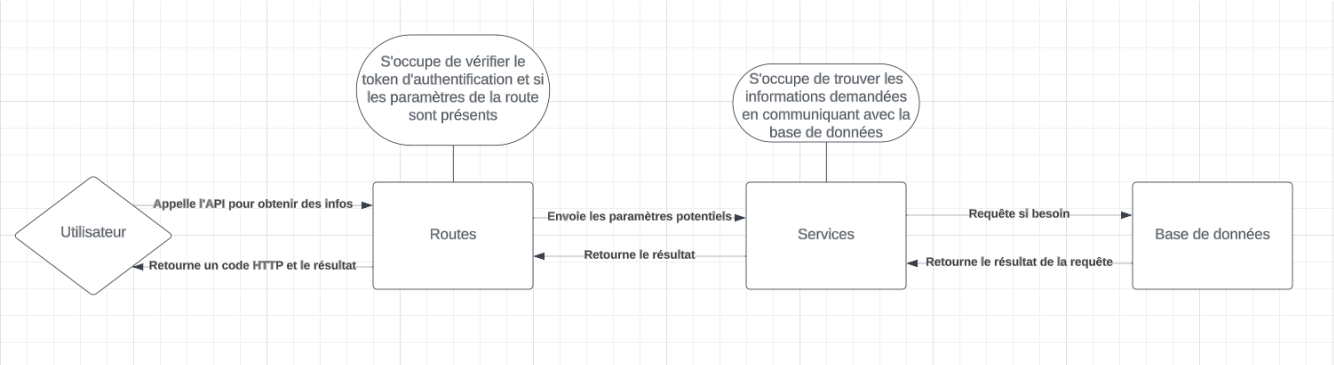
10.1. Hashing du wallet

Les wallets sont hashés et encryptés en SHA-512 et salés 100 fois avec une clé privée enregistrée dans une variable d'environnement sur le serveur afin de garantir sa sécurité.

10.2. Encodage du JWT

Le JWT est encodé (et décodé) à l'aide d'une clé privée enregistrée dans une variable d'environnement. L'algorithme utilisé est le HS256. Ce dernier est également enregistré dans une variable d'environnement.

11. Architecture du backend



12. Modèles du backend

Les modèles du backend sont utilisés pour les routes du backend que ce soit en entrée comme en sortie. Pour créer un modèle d'entrée, il suffit de le créer dans le dossier src/models/input. Pour un modèle de réponse, il faut le créer dans le dossier src/models/response.

Les modèles étant affichés dans les routes du backend, merci de vous référer aux différents points secondaires du point 13.

13. Routes du backend

13.1. Les sorties d'erreur universelles

13.1.1. La sortie code 401

La sortie de code 401 se produit lorsque l'utilisateur n'a pas la permission d'accéder à cette route, ou qu'il n'est pas authentifié. Le retour est donc de type PlainText, le body est donc une simple string

```
"Invalid token scheme")
```

13.1.2. La sortie code 422

La sortie de code 422 se produit automatiquement par le framework FastAPI lorsque l'utilisateur s'est trompé dans la saisie des paramètres. Cela peut survenir donc à cause d'un ajout d'un paramètre non demandé ou l'oubli d'un paramètre obligatoire. Il retourne un body de type application/json dont le contenu doit correspondre à ceci:

```
{
  "detail": [
    {
      "loc": [
        "string"
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```


13.1.3. La sortie code 500

La sortie de code 500 se produit lorsqu'il y a eu une erreur sur le serveur. Il retourne un body de type PlainText

13.2. Routes user

13.2.1. POST /login

Entrée

Body de type *application/json*


```
{
  "wallet_address": "string",
  "encrypted_wallet": "string",
  "token": "string"
}
```


Le token est nécessaire uniquement si la double authentification TOTP a été activée.

Sortie code 200

Body de type *application/json*


```
{
  "access_token": "string",
  "token_type": "string",
  "refresh_token": "string"
}
```


Cette sortie se produit lorsque l'utilisateur s'est bien connecté et est déjà inscrit.

Sortie code 201

Body de type *application/json*


```
{
  "access_token": "string",
  "token_type": "string",
  "refresh_token": "string"
}
```

Cette sortie se produit lorsque l'utilisateur s'est bien connecté et n'était pas inscrit.

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

Cette sortie se produit lorsque l'utilisateur s'est trompé sur la saisie des paramètres.

Sortie code 403

Body de type *application/json*

```
{
  "reason": "string",
  "expires": -1
}
```

Cette sortie se produit lorsque l'utilisateur a été banni de la plateforme.

13.2.2. POST /refresh_token

Entrée

Body de type application/json

```
{
  "refresh_token": "string"
}
```

Sortie code 200

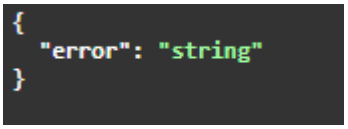
Body de type application/json

```
{
  "access_token": "string",
  "token_type": "string",
  "refresh_token": "string"
}
```

Cette sortie se produit lorsque l'utilisateur a bien réussi à générer un nouveau jeton d'authentification avec son jeton de rafraichissement.

Sortie code 400

Body de type application/json



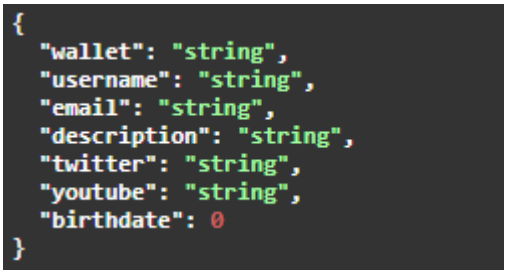
Cette sortie se produit lorsque l'utilisateur s'est trompé sur la valeur du jeton de rafraichissement.

13.2.3. GET /user/profile

Authentification JWTChecker nécessaire

Sortie code 200

Body de type application/json



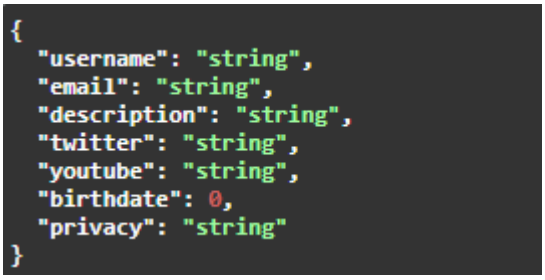
Cette sortie se produit lorsque l'utilisateur a bien réussi à récupérer son profil utilisateur sur l'application.

13.2.4. PATCH /user/profile

Authentification JWTChecker nécessaire

Entrée

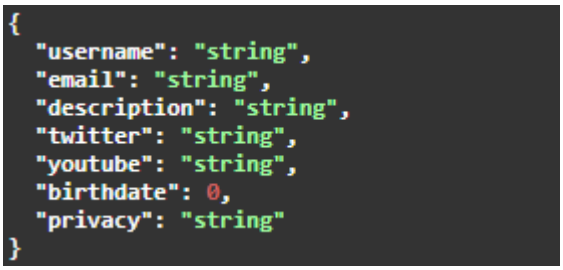
Body de type application/json



Aucun des paramètres n'est obligatoire, chaque paramètre saisi sera pris en compte dans la requête et sera modifiée par la valeur saisie en cas de réussite.

Sortie code 200

Body de type application/json



Cette sortie se produit lorsque l'utilisateur a bien modifié le champ souhaité. Tous les champs non modifiés seront retournés avec la valeur null

13.2.5. GET /user/profile/{username}

Authentification JWTChecker nécessaire

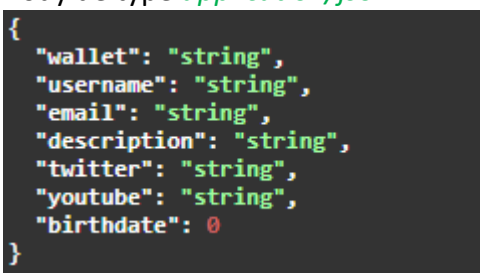
Entrée

Body de type path

Username => String

Sortie code 200

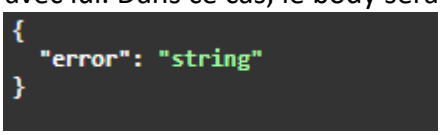
Body de type *application/json*



Cette sortie se produit lorsque l'utilisateur a bien pu récupérer le profil de l'utilisateur demandé dans la requête.

Sortie de code 401 (exceptionnel)

En plus de la sortie 401 possible à cause de l'échec de l'authentification, l'utilisateur peut être confronté à cette sortie si l'utilisateur qu'il recherche n'a pas souhaité partager ses informations avec lui. Dans ce cas, le body sera de type *application/json* et ressemblera à ceci:



13.2.6. POST /user/authenticator/qrcode

Authentification JWTChecker nécessaire

Entrée

Body de type *application/json*

```
{
  "wordlist": "string"
}
```

La wordlist est obligatoire uniquement si l'utilisateur a déjà utilisé la double authentification sur l'application.

Sortie code 200

Body de type *application/json*

```
{
  "qr": "string",
  "wordlist": "string"
}
```

Cette sortie se produit lorsque l'authentification à double facteur a bien été activé. Le lien a afficher sous forme de QR Code est envoyé ainsi qu'une wordlist s'il s'agit de la première fois que l'utilisateur utilise l'authentification à double facteur. Dans le cas contraire, la valeur de wordlist sera null

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

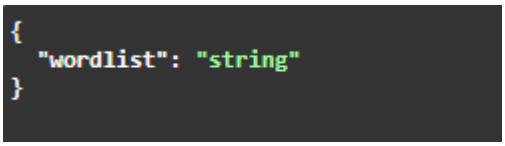
Cette sortie se produit si l'utilisateur n'a pas donné la bonne wordlist si ce dernier en avait une à fournir.

13.2.7. DELETE /user/authenticator/qrcode

Authentification JWTChecker nécessaire

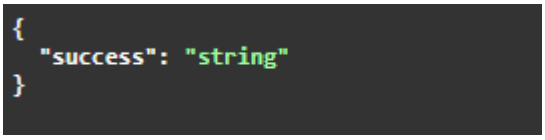
Entrée

Body de type *application/json*



Sortie code 200

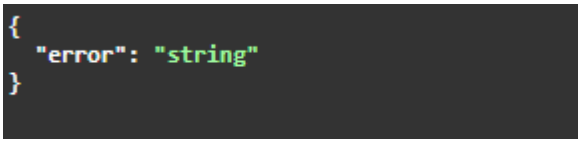
Body de type *application/json*



Cette sortie se produit lorsque l'utilisateur a bien supprimé la double authentification TOTP. Il faut tout de même penser à garder la wordlist qui elle n'est pas supprimée.

Sortie code 400

Body de type *application/json*



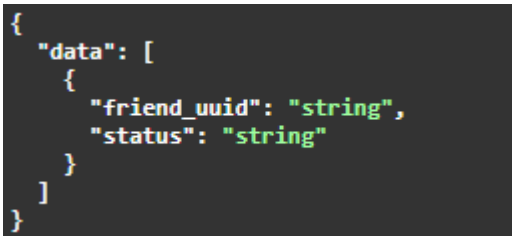
Cette sortie se produit lorsque l'utilisateur n'a pas renseigné la bonne wordlist.

13.2.8. GET /user/friends

Authentification JWTChecker nécessaire

Sortie code 200

Body de type *application/json*



Cette sortie se produit lorsque l'utilisateur a souhaité voir sa liste d'amis ainsi que les requêtes d'amis qu'il a reçu ou envoyé.

13.2.9. POST /user/friends

Authentification JWTChecker nécessaire

Entrée

Body de type *application/json*

```
{
  "friend_uuid": "string"
}
```

Sortie code 200

Body de type *application/json*

```
{
  "success": "string"
}
```

Cette sortie se produit lorsque l'utilisateur a bien accepté, ou envoyé la demande d'ami à l'utilisateur souhaité.

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

Cette sortie se produit lorsque l'utilisateur a déjà fait une demande d'ami ou est déjà ami avec l'utilisateur souhaité.

13.2.10. DELETE /user/friends

Authentification JWTChecker nécessaire

Entrée

Body de type *application/json*

```
{
  "friend_uuid": "string"
}
```

Sortie code 200

Body de type *application/json*

```
{
  "success": "string"
}
```

Cette sortie se produit lorsque l'utilisateur a bien supprimé l'amitié ou sa demande d'ami envers l'utilisateur souhaité.

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

Cette sortie se produit lorsque l'utilisateur n'a pas fait de demande ou n'est pas ami avec l'utilisateur souhaité.

13.2.11. GET /user/search

Authentification JWTChecker nécessaire

Entrée

Type *query params*

Sortie code 200

Body de type *application/json*

```
{
  "total": 10,
  "datas": [
    {
      "uuid": "0a7c2351-b004-4ef7-9500-c5e85bce77d7",
      "username": "0xf1F3Ad462941E9d1C615E405C05B75516B61fbb0"
    },
```


Cette sortie se produit lorsque l'utilisateur recherche un utilisateur ayant pour nom la valeur de user ou contenant la valeur de user dans son nom.

Sortie code 400

Body de type *application/json*


```
{
  "error": "You must provide a partial username"
}
```


Cette sortie se produit lorsque l'utilisateur ne rentre pas correctement les valeurs pour faire une recherche (saisie par exemple de page = -1).

13.3. Routes Tutorials

13.3.1. GET /tuto/all

Sortie code 200

Body de type *application/json*


```
{
  "data": [
    {
      "id": 0,
      "title": "string",
      "markdownUrl": "string",
      "category": "string",
      "answer": "string",
      "startCode": "string",
      "shouldBeCheck": true,
      "enabled": true,
      "points": 0
    }
  ]
}
```


Cette sortie se produit lorsque l'utilisateur souhaite voir tous les tutoriels disponibles sur l'application.

13.3.2. GET /tuto/{id}

Entrée

Body de type *path*

ID => Number

Sortie code 200

Body de type *application/json*


```
{
  "id": 0,
  "title": "string",
  "markdownUrl": "string",
  "category": "string",
  "answer": "string",
  "startCode": "string",
  "shouldBeCheck": true,
  "enabled": true,
  "points": 0
}
```


Cette sortie se produit lorsque l'utilisateur consulte le tutoriel souhaité

Sortie code 400

Body de type *application/json*


```
{
  "error": "string"
}
```


Cette sortie se produit lorsque l'utilisateur renseigne un id de tutoriel invalide.

13.3.3. GET/tuto/category/all

Sortie code 200

Body de type *application/json*


```
{
  "data": [
    {
      "name": "string",
      "description": "string",
      "image": "string"
    }
  ]
}
```

Cette sortie se produit lorsque l'utilisateur souhaite voir l'ensemble des catégories existantes sur l'application.

13.3.4. GET/tuto/category/{category}

Entrée

Body de type path

Category => String

Sortie code 200

Body de type *application/json*

```
{
  "data": [
    {
      "id": 0,
      "title": "string",
      "markdownUrl": "string",
      "category": "string",
      "answer": "string",
      "startCode": "string",
      "shouldBeCheck": true,
      "enabled": true,
      "points": 0
    }
  ]
}
```

Cette sortie se produit lorsque l'utilisateur souhaite consulter la catégorie souhaitée.

13.3.5. GET tuto/scoreboard/id/{id}

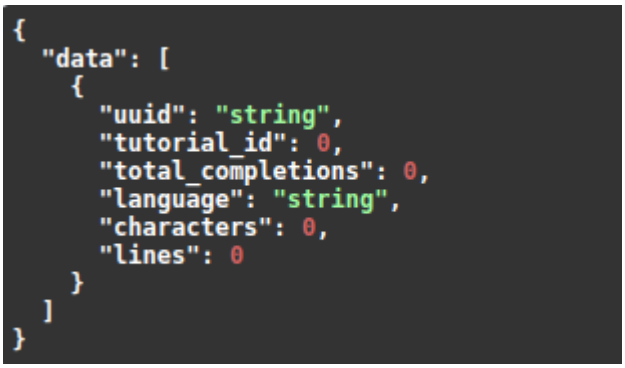
Entrée

Body de type path

ID => Number

Sortie code 200

Body de type *application/json*



```
{
  "data": [
    {
      "uuid": "string",
      "tutorial_id": 0,
      "total_completions": 0,
      "language": "string",
      "characters": 0,
      "lines": 0
    }
  ]
}
```


Cette sortie se produit lorsque l'utilisateur souhaite voir la liste et le classement des utilisateurs ayant réussi le tutoriel souhaité.

13.3.6. GET tuto/success/id/{id}

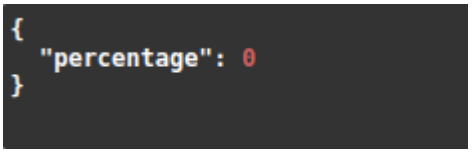
Entrée

Body de type path

ID => Number

Sortie code 200

Body de type *application/json*



```
{
  "percentage": 0
}
```


Cette sortie se produit lorsque l'utilisateur souhaite voir le pourcentage d'utilisateurs ayant réussi le tutoriel en fonction du nombre d'utilisateurs inscrits.

13.3.7. GET /tuto/scoreboard/me

Authentification JWTChecker nécessaire

Sortie code 200

Body de type *application/json*

```
{
  "data": [
    {
      "uuid": "string",
      "tutorial_id": 0,
      "total_completions": 0,
      "language": "string",
      "characters": 0,
      "lines": 0
    }
  ]
}
```


Cette sortie se produit lorsque l'utilisateur souhaite consulter ses scores sur tous les tutoriels qu'il a accompli.

13.3.8. GET /tuto/success/me

Authentification JWTChecker nécessaire

Sortie code 200

Body de type *application/json*

```
{
  "data": [
    {
      "uuid": "string",
      "tutorial_id": 0,
      "total_completions": 0,
      "language": "string",
      "characters": 0,
      "lines": 0
    }
  ],
  "total_completion": 0
}
```


Cette sortie se produit lorsque l'utilisateur souhaite voir son meilleur score sur les tutoriaux ainsi que son nombre de complétions.

13.3.9. POST /tuto/complete

Authentification JWTChecker nécessaire

Entrée

Body de type *application/json*

```
{
  "source_code": "string",
  "tutorial_id": 0,
  "total_completions": 0,
  "language": "string",
  "characters": 0,
  "lines": 0,
  "exec": true
}
```


Sortie code 200

Body de type *application/json*


```
{
  "is_correct": true,
  "total_completions": 0,
  "error_description": "string"
}
```


Cette sortie se produit lorsque l'utilisateur a envoyé sa réponse à un tutoriel. Il sera correct ou incorrect. Dans le cas où le tutoriel est incorrect, is_correct sera a false et si une erreur dans son code est trouvé, elle sera donnée via error_description.

Sortie code 400

Body de type *application/json*


```
{
  "error": "string"
}
```


Cette sortie se produit lorsque l'utilisateur souhaite envoyer la réponse d'un tutoriel inexistant.

13.4. Routes Article

13.4.1. GET /article/all

Sortie code 200

Body de type *application/json*

```
{
  "data": [
    {
      "id": 0,
      "title": "string",
      "markdownUrl": "string",
      "shortDescription": "string",
      "publicationDate": 0,
      "author": "string"
    }
  ]
}
```


Cette sortie se produit lorsque l'utilisateur souhaite consulter tous les articles du blog disponibles.

13.4.2. GET /article/id/{id}

Entrée

Body de type path

ID => Number

Sortie code 200

Body de type *application/json*


```
{
  "id": 0,
  "title": "string",
  "markdownUrl": "string",
  "shortDescription": "string",
  "publicationDate": 0,
  "author": "string"
}
```


Cette sortie se produit lorsque l'utilisateur souhaite consulter un article du blog.

13.4.3. POST /article/create

AdminChecker niveau 1 nécessaire

Entrée

Body de type *application/json*


```
{
  "id": -1,
  "title": "string",
  "markdownUrl": "string",
  "author": "string",
  "shortDescription": "string"
}
```

Sortie code 201

Body de type *application/json*

```
{
  "success": "string"
}
```

Cette sortie se produit lorsque l'administrateur ou le modérateur a créé un article.

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

Cette sortie se produit lorsque l'administrateur ou le modérateur a voulu créer un article sur un id déjà existant.

13.4.4. PATCH article/update

AdminChecker niveau 1 nécessaire

Entrée

Body de type *application/json*

```
{
  "id": -1,
  "title": "string",
  "markdownUrl": "string",
  "author": "string",
  "shortDescription": "string"
}
```

Sortie code 200

Body de type *application/json*

```
{
  "id": 0,
  "title": "string",
  "markdownUrl": "string",
  "shortDescription": "string",
  "publicationDate": 0,
  "author": "string"
}
```

Cette sortie se produit lorsque l'administrateur ou le modérateur a modifié un article

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

Cette sortie se produit lorsque l'administrateur ou le modérateur a souhaité modifier un article inexistant.

13.4.5. DELETE article/delete

AdminChecker niveau 1 nécessaire

Entrée

Body de type *application/json*

```
{
  "id": 0
}
```

Sortie code 200

Body de type *application/json*

```
{
  "success": "string"
}
```

Cette sortie se produit lorsque l'administrateur ou le modérateur a bien supprimé l'article souhaité.

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

Cette sortie se produit lorsque l'administrateur ou le modérateur a souhaité supprimé un article inexistant.

13.5. Routes Moderation

13.5.1. GET admin/banlist/{uuid}

AdminChecker niveau 1 nécessaire

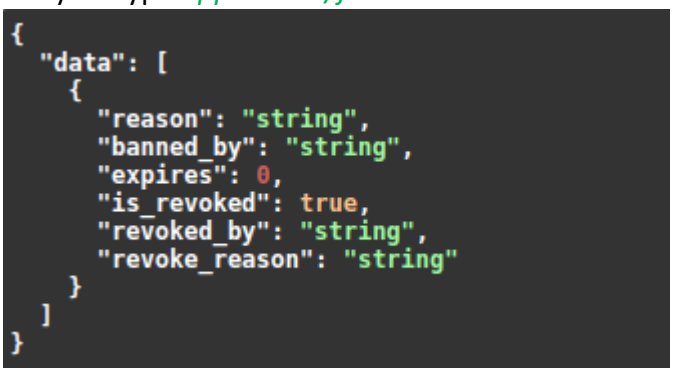
Entrée

Body de type path

ID => Number

Sortie code 200

Body de type *application/json*

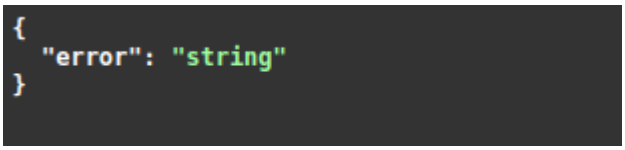


```
{
  "data": [
    {
      "reason": "string",
      "banned_by": "string",
      "expires": 0,
      "is_revoked": true,
      "revoked_by": "string",
      "revoke_reason": "string"
    }
  ]
}
```

Cette sortie se produit lorsque l'administrateur ou le modérateur a voulu consulter l'historique des sanctions d'un utilisateur souhaité.

Sortie code 400

Body de type *application/json*



```
{
  "error": "string"
}
```

Cette sortie se produit lorsque l'administrateur ou le modérateur a voulu consulter l'historique des sanctions d'un utilisateur inexistant.

13.5.2. POST admin/ban

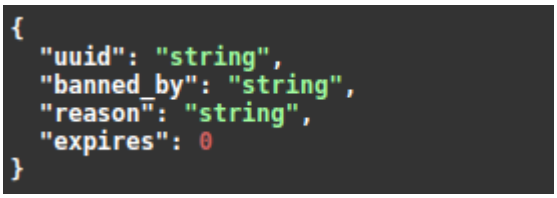
AdminChecker niveau 1 nécessaire

Entrée

Body de type *application/json*

Sortie code 200

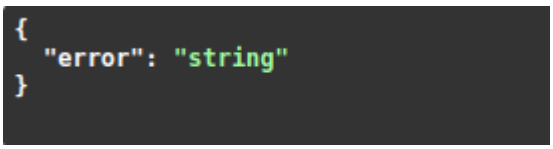
Body de type *application/json*



Cette sortie se produit lorsque l'administrateur ou le modérateur a banni un utilisateur souhaité. Si la valeur de expires est de –1, le bannissement est définitif.

Sortie code 400

Body de type *application/json*



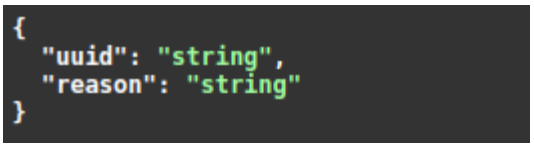
Cette sortie se produit lorsque l'administrateur ou le modérateur a voulu bannir un utilisateur non existant ou déjà banni.

13.5.3. POST admin/unban

AdminChecker niveau 1 nécessaire

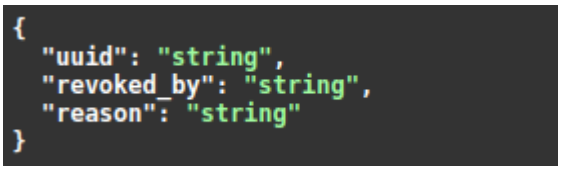
Entrée

Body de type *application/json*



Sortie code 200

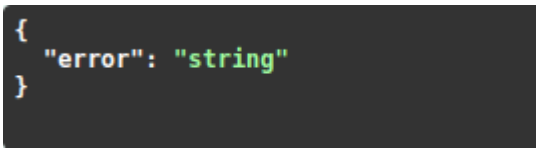
Body de type *application/json*



Cette sortie se produit lorsque l'administrateur ou le modérateur a débanni un utilisateur souhaité.

Sortie code 400

Body de type *application/json*



Cette sortie se produit lorsque l'administrateur ou le modérateur a souhaité débannir un utilisateur qui n'existe pas ou qui n'est pas banni.

13.6. Routes FAQ

13.6.1. GET faq/all

Sortie code 200

Body de type *application/json*

Cette sortie se produit lorsque l'utilisateur souhaite consulter les questions disponibles dans la FAQ.

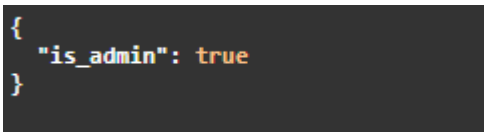
13.7. Routes Admin

13.7.1. GET /admin/is_admin

Authentification JWTChecker nécessaire

Sortie code 200

Body de type *application/json*



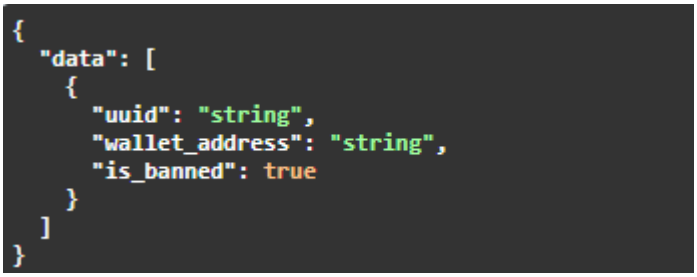
Cette sortie se produit lorsque l'utilisateur a pu vérifier son statut sur l'application. Retourne true si l'utilisateur est administrateur, dans le cas contraire il retourne false.

13.7.2. GET /admin/users

Authentification AdminChecker de niveau 2 nécessaire

Sortie de code 200

Body de type *application/json*



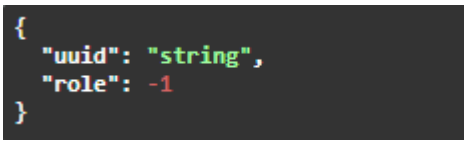
Cette sortie se produit lorsque l'administrateur souhaite voir la liste de tous les utilisateurs présents dans l'application.

13.7.3. POST /admin/mod

Authentification AdminChecker de niveau 2 nécessaire

Entrée

Body de type *application/json*



Sortie code 200

Body de type *application/json*

```
{
  "uuid": "string",
  "role": -1
}
```

Cette sortie se produit lorsque l'administrateur a ajouté ou supprimé les droits de modération ou d'administration a un utilisateur. Si le role est –1, il est supprimé, si le role est 1, l'utilisateur est modérateur, si le role est 2, l'utilisateur est administrateur.

13.7.4. POST /admin/tuto/create

Authentification AdminChecker de niveau 2 nécessaire

Entrée

Body de type *application/json*

```
{
  "title": "string",
  "markdownUrl": "string",
  "category": "string",
  "answer": "string",
  "startCode": "string",
  "shouldBeCheck": true,
  "input": "string",
  "points": 0
}
```

Sortie code 200

Body de type *application/json*

```
{
  "success": "string"
}
```

Cette sortie se produit lorsque l'administrateur a bien réussi à créer un tutoriel sur l'application.

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

Cette sortie se produit lorsque l'administrateur n'a pas correctement saisi les données pour créer un tutoriel. L'erreur peut venir de n'importe quel champ, renseigner un nombre de points négatif comme une mauvaise catégorie.

13.7.5. POST /admin/tuto/update

Authentification AdminChecker de niveau 2 nécessaire

Entrée

Body de type *application/json*

```
{
  "id": 0,
  "title": "string",
  "markdownUrl": "string",
  "category": "string",
  "answer": "string",
  "startCode": "string",
  "shouldBeCheck": true,
  "input": "string",
  "points": 0
}
```

Sortie code 200

Body de type *application/json*

```
{
  "success": "string"
}
```

Cette sortie se produit lorsque le tutoriel a bien été modifié. Attention à bien renseigner tout les champs, et ne pas faire comme dans la route de modification des informations utilisateur.

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

Cette sortie se produit notamment si l'id du tutoriel est invalide.

13.7.6. PATCH /admin/tuto/toggle

Authentification AdminChecker de niveau 2 nécessaire

Entrée

Body de type *application/json*

```
{
  "id": 0
}
```

Sortie code 200

Body de type *application/json*

```
{
  "id": 0,
  "enabled": true
}
```

Cette sortie se produit lorsque l'utilisateur active ou désactive un tutoriel.

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

Cette sortie se produit lorsque l'utilisateur s'est trompé sur l'ID du tutoriel.

13.7.7. POST /admin/category/create

Authentification AdminChecker de niveau 2 nécessaire

Entrée

Body de type *application/json*

```
{
  "name": "string",
  "description": "string",
  "image": "string"
}
```

Sortie code 200

Body de type *application/json*

```
{
  "success": "string"
}
```

Cette sortie se produit lorsque l'administrateur a bien réussi à créer une nouvelle catégorie

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

Cette sortie se produit lorsque l'administrateur a essayé de créer une catégorie portant déjà le nom d'une catégorie déjà existante.

13.7.8. PATCH /admin/category/update

Authentification AdminChecker de niveau 2 nécessaire

Entrée

Body de type *application/json*

```
{
  "name": "string",
  "description": "string",
  "image": "string"
}
```

Sortie code 200

Body de type *application/json*

```
{
  "success": "string"
}
```

Cette sortie se produit lorsque l'administrateur a bien modifié la catégorie (il a changé la description et/ou l'image)

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

Cette sortie se produit lorsque l'administrateur a voulu modifier une catégorie qui n'existe pas.

13.7.9. DELETE /admin/category/delete

Authentification AdminChecker de niveau 2 nécessaire

Entrée
Body de type *application/json*

```
{
  "name": "string"
}
```

Sortie code 200

Body de type *application/json*

```
{
  "success": "string"
}
```

Cette sortie se produit lorsque l'administrateur a bien réussi à supprimer la catégorie

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

Cette sortie se produit lorsque l'administrateur a souhaité supprimer une catégorie qui n'existe pas.

13.7.10. POST admin/article/create_markdown

Authentification AdminChecker de niveau 2 nécessaire

Entrée

Body de type *application/json*

```
{
  "name": "string",
  "content": "string"
}
```

Sortie code 201

Body de type *application/json*

```
{
  "success": "string",
  "markdown_url": "string"
}
```


Cette sortie se produit lorsque l'administrateur crée un fichier markdown.

Sortie code 400

Body de type *application/json*


```
{
  "error": "string"
}
```


Cette sortie se produit lorsque l'administrateur souhaite créer un fichier markdown avec un nom déjà existant.

13.7.11. GET admin/article/available_markdown

Authentification AdminChecker de niveau 2 nécessaire

Sortie code 200

Body de type *application/json*


```
{
  "success": "string",
  "markdowns": [
    {
      "title": "string",
      "markdown_url": "string"
    }
  ]
}
```


Cette sortie se produit lorsque l'administrateur souhaite consulter la liste des markdown disponibles pour un article, ainsi que son titre.

Sortie code 400

Body de type *application/json*


```
{
  "error": "string"
}
```


Cette sortie se produit lorsque l'administrateur n'a pas pu récupérer la liste des markdown disponibles.

13.7.12. POST admin/tuto/create_markdown

Authentification AdminChecker de niveau 2 nécessaire

Entrée

Body de type *application/json*

```
{
  "name": "string",
  "content": "string"
}
```

Sortie code 201

Body de type *application/json*

```
{
  "success": "string",
  "markdown_url": "string"
}
```

Cette sortie se produit lorsque l'administrateur a réussi à créer un fichier markdown pour les tutoriels.

Sortie code 400

Body de type *application/json*

```
{
  "error": "string"
}
```

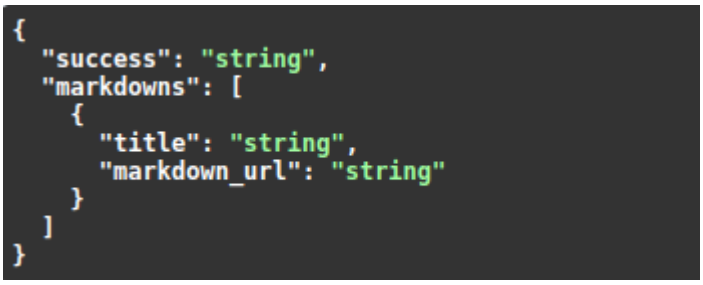
Cette sortie se produit lorsque l'administrateur a voulu créer un fichier markdown pour les tutoriels mais qu'un fichier du même nom existe déjà.

13.7.13. GET admin/tuto/available_markdown

Authentification AdminChecker de niveau 2 nécessaire

Sortie code 200

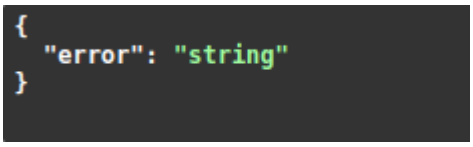
Body de type *application/json*



Cette sortie se produit lorsque l'administrateur souhaite consulter la liste des fichiers markdown disponibles pour les tutoriels.

Sortie code 400

Body de type *application/json*



Cette sortie se produit lorsque l'administrateur n'a pas pu accéder à la liste des fichiers markdown disponibles pour les tutoriels.

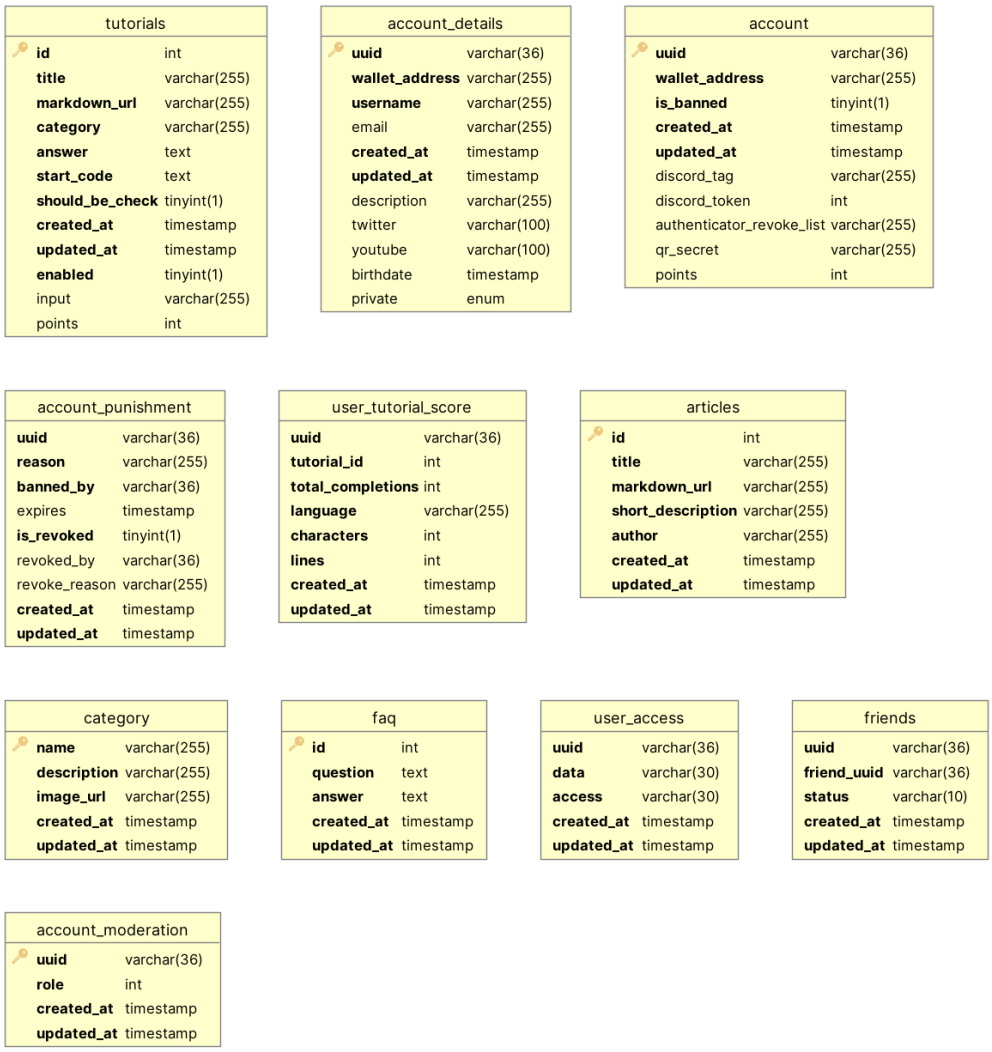
14. CODE_EXECUTER

14.1. Ajouter une prise en charge d'un nouveau langage

Pour prendre en charge un nouveau langage, il vous faudra créer un nouveau fichier dans src/langagues/
Nommez-le avec le nom du langage que vous ajoutez, codez l'ajout de langage et ensuite dans index.js, rajoutez le langage ajouté (ex : js, c, cpp, R, py …) dans la variable « supportedLanguages », puis importez votre fonction et enfin dans « switch (language) { » rajoutez une case avec le langage que vous venez d'ajouter.

15. Base de données

15.1. Diagramme



Il existe des contraintes d'unicité dans les tables suivantes:

- user_tutorial_score : (uuid, tutorial_id, language)
- user_access: (uuid, data), index sur uuid.
- friends: (uuid, friend_uuid), index sur uuid.

15.2. Informations générales

Pour la base de données, nous utilisons une base de données SQL, ici MySQL.

15.3. Gestion de la base de données

La base de données est gérée dans le même dossier que le backend. Le backend étant la seule partie à avoir une liaison directe avec la base de données, cette dernière est gérée via plusieurs petits scripts bash.

15.3.1. Créer, modifier ou supprimer une table

Afin de créer, modifier ou supprimer une table ou tout autre élément dans MySQL comme un trigger ou une fonction, il faut appeler le script “create_db_update.sh” présent dans le dossier "scripts” du backend. Ce script demandera à l’utilisateur le nom du fichier à créer et le fera dans le dossier “db”. Le fichier doit être précédé d’un nombre et être au format sql. A l’intérieur, il faudra écrire soi-même les modifications que l’on veut apporter dans la base de données de l’application.
Lors de la création du fichier sql, il y a un fichier “version” dans le dossier “db” contenant la version du dernier fichier créé afin de permettre au script de savoir le nombre à inscrire avant le nom du fichier.

15.3.2. Application de la modification de la base de données

L’application du fichier créé et de son contenu sera automatiquement fait lors de l’exécution du script “update_db.sh” dans le dossier “scripts”. Ce dernier est automatiquement appelé lorsque l’on souhaite lancer le backend via le script “start.sh” présent à la racine du dossier du backend.
La mise à jour se fait en plusieurs étapes:

- Vérification de la version actuelle de la base de données de l’utilisateur (récupérée d’une variable environnement)
- Vérification des fichiers présents dans le dossier “db”. Si le fichier n’est pas un fichier au format sql, il ne sera pas pris en compte.
- Triage des fichiers sql par numéro de version.
- Application des modifications contenues dans les fichiers correspondant à une version plus récente que celle de la base de données de l’utilisateur (dans l’ordre afin d’éviter au maximum les erreurs)
- Changement de la valeur de la version de la base de données dans la variable d’environnement.

Dans le cas où l’utilisateur a déjà la version la plus récente de la base de données, aucun fichier sql présent dans le dossier “db” n’est exécuté.

Il est préférable de lancer le backend via le script “start.sh” pour faire les mises à jour de la base de données.

16. Accéder à la documentation utilisateur

Pour accéder à la documentation utilisateur, vous pouvez aller sur ce lien: https://epitechfr-my.sharepoint.com/:w:/g/personal/gabriel_knies_epitech_eu/Ef_P-q4CbY1BumSSLEkveyoBA_2OpeLfvEp65bEZxKPoaA?e=NRuLKj