TP 8 : Blockchain et Application décentralisée DAPP

1. **Mise en place** de l' **environnement de développement**

Truffle est le framework de développement le plus populaire pour Ethereum, conçu pour vous simplifier la vie. Mais avant d'installer Truffle, assurez-vous d' installer Node.js.

Une fois Node installé, une seule commande suffit pour installer Truffle :

```
npm install -g truffle
```

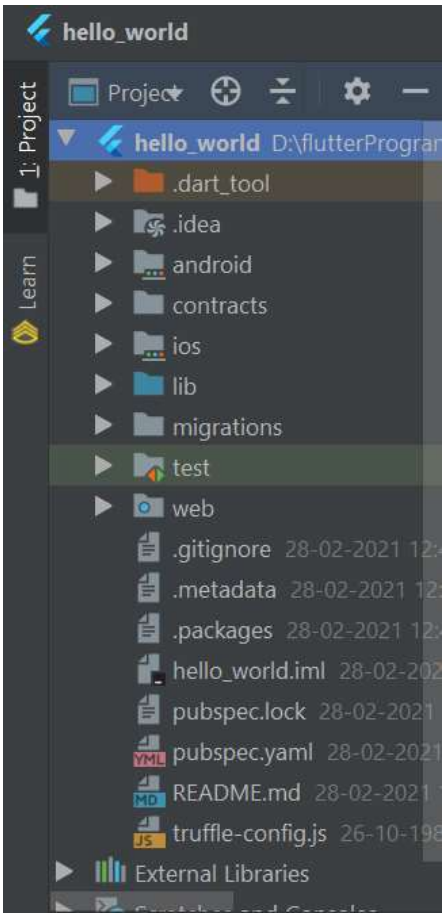
Nous utiliserons également Ganache , une blockchain personnelle pour le développement Ethereum permettant de déployer des contrats intelligents, de développer des applications et d'exécuter des tests. Vous pouvez télécharger Ganache en vous rendant sur https://archive.trufflesuite.com/ganache/ et en cliquant sur le bouton « Télécharger ».

1. **Création d'un projet de truffes**

1. Créez un projet Flutter de base dans votre IDE préféré.
2. Initialisez Truffle dans le répertoire du projet Flutter en exécutant la commande suivante :

```
truffle init
```

Structure du répertoire



- **contrats/** : Contient le fichier de contrat Solidity.

- **migrations/** : Contient les fichiers de script de migration (Truffle utilise un système de migration pour gérer le déploiement des contrats).
- **test/** : Contient les fichiers de script de test.
- **truffle-config.js** : Contient les informations de configuration du déploiement de Truffle.

3. Rédiger votre premier contrat intelligent

Le contrat intelligent fait en réalité office de logique back-end et de stockage pour notre application décentralisée.

1. Créez un nouveau fichier nommé ***HelloWorld.sol*** dans le répertoire **contracts/ .**
2. Ajoutez le contenu suivant au fichier :

```
pragma solidity ^0.5.9;

contract HelloWorld {

}
```

- La version minimale de Solidity requise est indiquée en haut du contrat : **pragma solidity ^0.5.9;** .
- Les instructions se terminent par un point-virgule.

4. Configuration variable

1. Ajoutez la variable suivante sur la ligne suivante après **le contrat HelloWorld {**

```
string public yourName
```

Nous venons de définir une seule variable ***yourName*** de type **chaîne de caractères** , et ***yourName*** est un modificateur **public** , ce qui signifie que nous pouvons y accéder depuis l'extérieur du contrat intelligent.

5. Constructeur

1. Ajoutez le constructeur suivant sur la ligne suivante après **la chaîne public yourName ;**

```
constructor() public {
    yourName = "Unknown" ;
}
```

Le constructeur Solidity n'est exécuté qu'une seule fois, lors de la création d'un contrat, et sert à initialiser l'état de ce dernier. Ici, nous attribuons simplement la valeur « Inconnu » à la variable ` **yourName**` .

6. Fonction

1. Ajoutez la fonction suivante au contrat intelligent après la déclaration du constructeur que nous avons définie ci-dessus.

```
function setName(string memory nm) public{
    yourName = nm ;
}
```

- Dans la fonction ci-dessus, nous allons prendre en entrée un ***nm*** (*chaîne de caractères*) et affecter cette valeur à la variable ***yourName*** .
- **La mémoire** est l'emplacement des données.

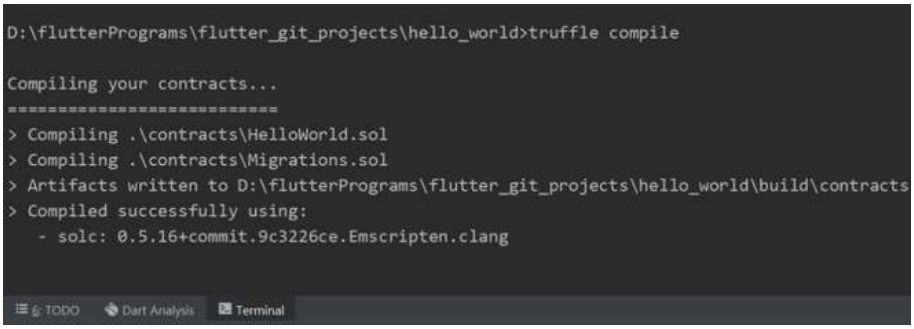
7. Compilation et migration

▪ Compilation

1. Dans un terminal, assurez-vous d'être à la racine du répertoire contenant le projet Flutter et Truffle, puis exécutez la commande suivante :

```
truffle compile
```

Vous devriez obtenir un résultat similaire à celui-ci :



truffes compiler

▪ Migration

Vous trouverez déjà un fichier JavaScript dans le répertoire **migrations/ : *1_initial_migration.js*** . Ce fichier gère le déploiement du contrat **Migrations.sol** pour observer les migrations ultérieures des contrats intelligents et garantit que nous n'effectuerons pas de double migration des contrats inchangés à l'avenir.

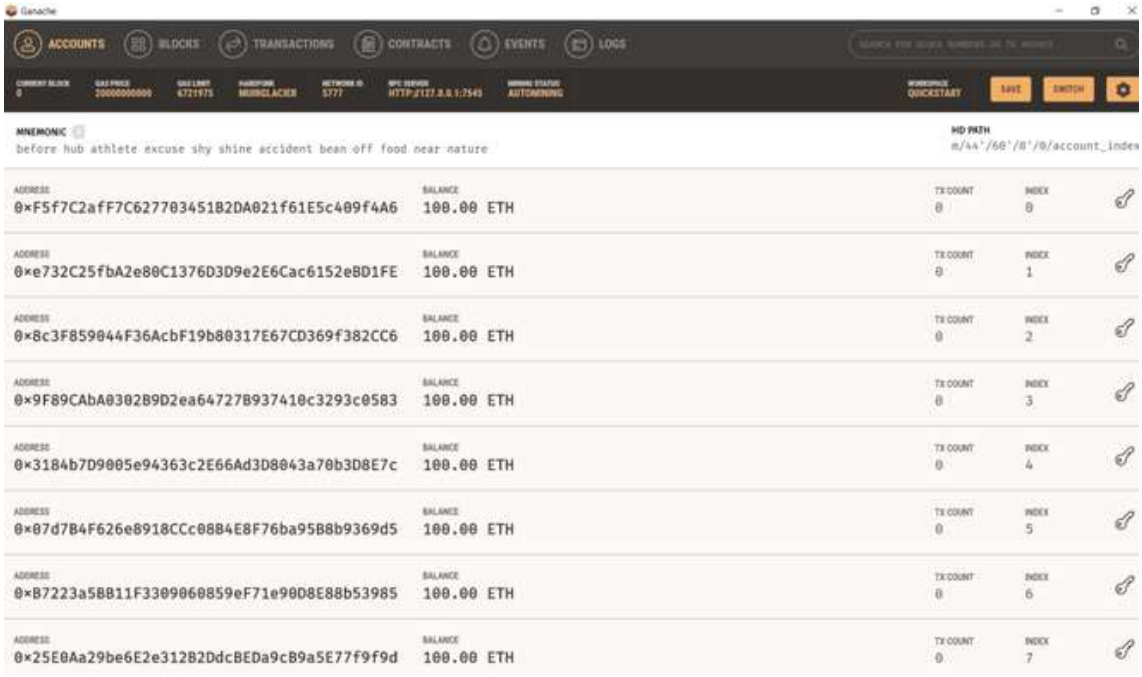
▪ Créons notre propre script de migration :

1. Créez un nouveau fichier nommé ***2_deploy_contracts.js*** dans le répertoire **migrations/ .**
2. Ajoutez le contenu suivant au fichier ***2_deploy_contracts.js*** :

```
const HelloWorld = artifacts.require("HelloWorld");

module.exports = function (deployer) {
  deployer.deploy(HelloWorld);
};
```

- Avant de pouvoir migrer notre contrat vers la blockchain, il nous faut une blockchain opérationnelle. Dans cet article, nous utiliserons **Ganache** , une blockchain personnelle pour le développement Ethereum permettant de déployer des contrats, de développer des

applications et d'exécuter des tests. Si ce n'est pas déjà fait, téléchargez **Ganache** et double-cliquez sur son icône pour lancer l'application. Une blockchain locale sera alors créée et exécutée sur le port 7545.



Ganache

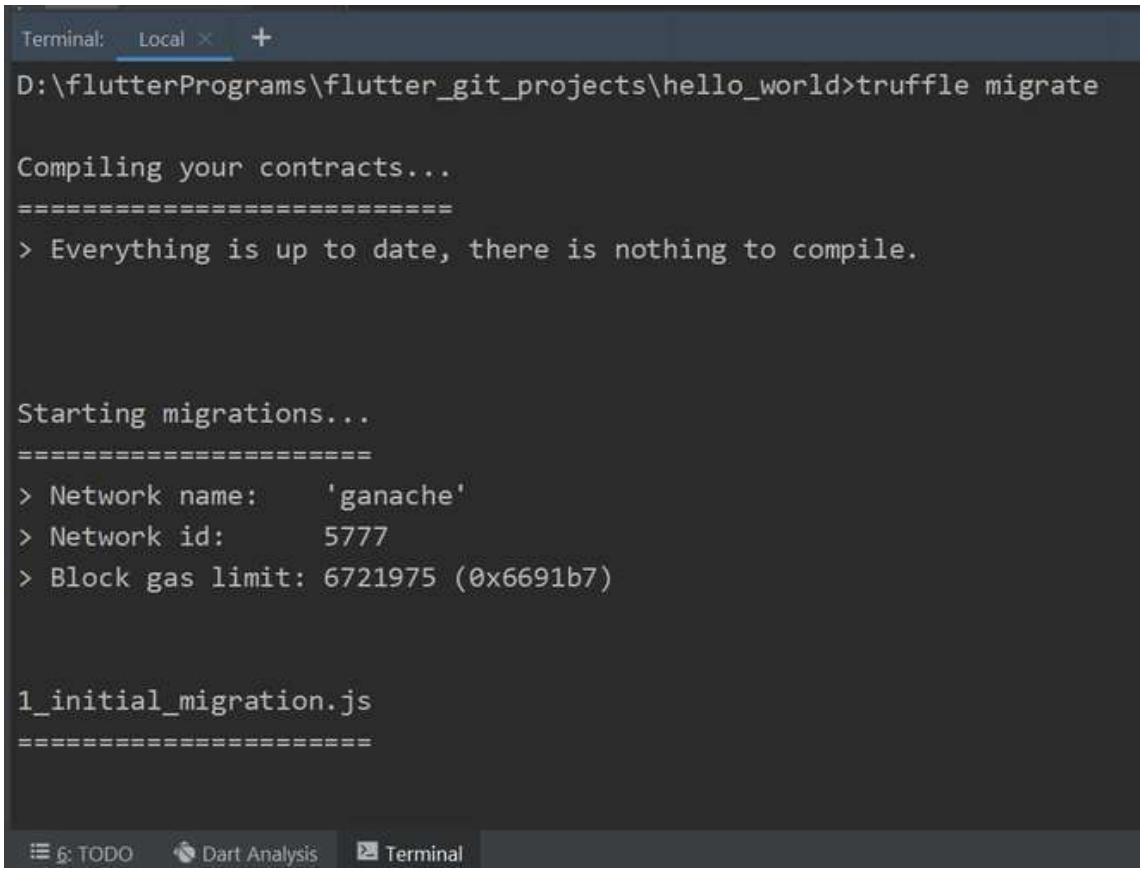
- Ajoutez le contenu suivant au fichier ***truffle-config.js*** :

```
module.exports = {
 networks: {
  development: {
     host: "127.0.0.1",     // Localhost (default: none)
     port: 8545,            // Standard Ethereum port (default: none)
     network_id: "*",       // Any network (default: none)
  },
 },
  contracts_build_directory: "./src/artifacts/",

 // Configure your compilers
 compilers: {
  solc: {

     // See the solidity docs for advice
     // about optimization and evmVersion
       optimizer: {
          enabled: false,
          runs: 200
       },
       version: "0.8.21",
       evmVersion: "byzantium"
     }
   }
};
```

- Pour migrer le contrat vers la blockchain, exécutez :

```
truffle migrate
```

Vous devriez obtenir un résultat similaire à celui-ci :



migration de la truffe

- Jetez un œil à Ganache : le premier compte disposait initialement de 100 ethers ; ce montant est désormais inférieur en raison des coûts de transaction liés à la migration.

8. Test du contrat intelligent

Dans Truffle, nous pouvons écrire des tests en JavaScript ou en Solidity. Dans ce TP, nous écrirons nos tests en JavaScript en utilisant les bibliothèques Chai et Mocha.

1. Créez un nouveau fichier nommé ***helloWorld.js*** dans le répertoire **test/ .**
2. Ajoutez le contenu suivant au fichier ***helloWorld.js*** :

```
const HelloWorld = artifacts.require("HelloWorld") ;
contract("HelloWorld" , () => {
   it("Hello World Testing" , async () => {
      const helloWorld = await HelloWorld.deployed() ;
      await helloWorld.setName("User Name") ;
      const result = await helloWorld.yourName() ;
      assert(result === "User Name") ;
   });
});
```

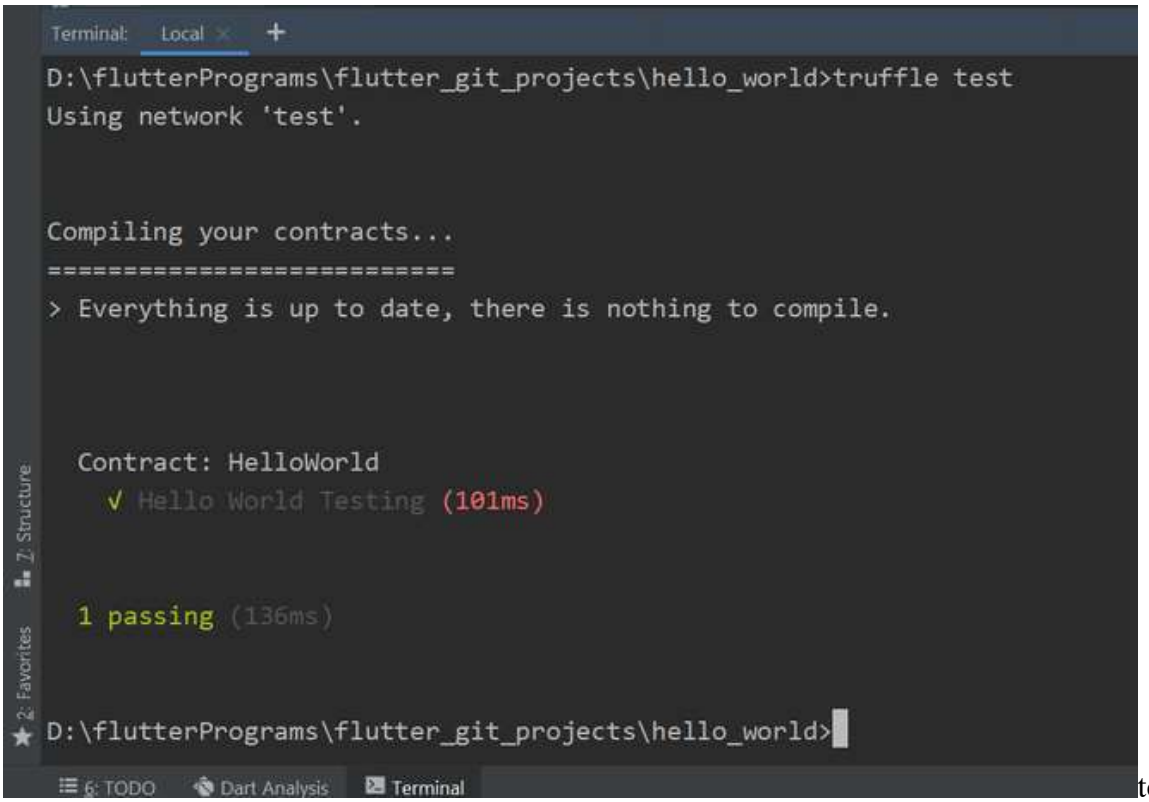
- **HelloWorld** : Le contrat intelligent que nous voulons tester. Nous commençons notre test en important notre contrat ***HelloWorld*** **à l'aide de artifacts.require** .
- Pour tester la fonction **setName** , rappelez-vous qu'elle accepte un **nom** (chaîne de caractères) comme argument.
- De plus, la variable **yourName** dans notre contrat utilise le modificateur **public** , que nous pouvons utiliser comme getter depuis une fonction externe.
- Truffle importe Chai, ce qui nous permet d'utiliser la fonction assert. Nous passons la valeur réelle et la valeur attendue. Pour vérifier que le nom est correctement défini ou non, **assert(result === "User Name") ;** .

9. Exécution des tests

- Exécution du test :

```
truffle test
```

- Si tous les tests réussissent, vous verrez un résultat dans la console similaire à celui-ci :



test de truffe

10. Liaison de contrats avec Flutter

- Dans le fichier ***pubspec.yaml,*** importez les packages suivants :

```
provider: ^4.3.3
web3dart: ^1.2.3
http: ^0.12.2
web_socket_channel: ^1.2.0
```

- Ajoutez également le fichier asset ***src/artifacts/HelloWorld.json*** au fichier **pubspec.yaml** généré par truffle-config.js lors de la migration de notre contrat.

```
assets:
    - src/artifacts/HelloWorld.json
```

1. Créez un nouveau fichier nommé ***contract_linking.dart*** dans le répertoire **lib/ .**
2. Ajoutez le contenu suivant au fichier :

```
import 'package:flutter/material.dart';

class ContractLinking extends ChangeNotifier {


}
```

- Une simple classe avec **ChangeNotifier** pour la gestion d'état.

Variables

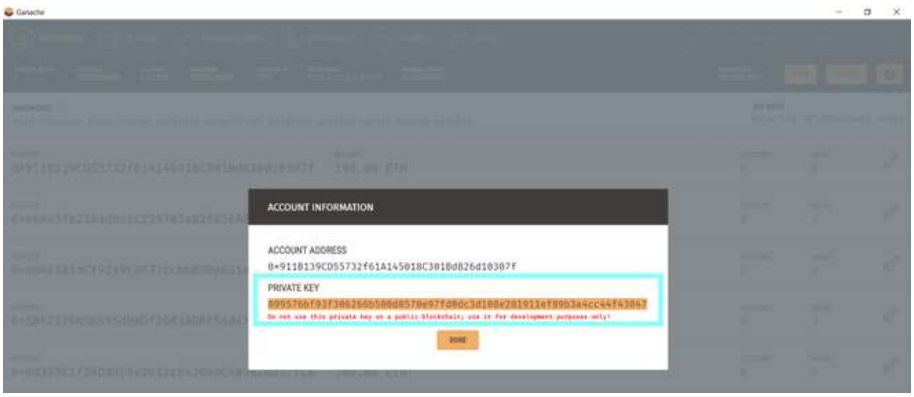
- Ajoutez la variable suivante sur la ligne suivante après **class ContractLinking extends ChangeNotifier { .**

```
final String _rpcUrl = "http://10.0.2.2:7545"

final String _wsUrl = "ws://10.0.2.2:7545/"

final String _privateKey = "Enter Private Key"
```

La bibliothèque **web3dart** n'envoie pas elle-même les transactions signées aux mineurs. Elle s'appuie sur un client RPC pour cela. Pour l'URL WebSocket, il suffit de modifier l'URL RPC. Vous pouvez obtenir cette URL depuis Ganache :

Ganache - URL RPC

- Récupérez la clé privée depuis Ganache :



Ganache - Clé privée

Si vous ne parvenez pas à trouver la clé privée , exécutez la commande ci-dessous dans le terminal .

```
npx ganache
```

- Déclarez les variables suivantes ci-dessous :

```
late Web3Client _client;

bool isLoading = true;

late String _abiCode;

late EthereumAddress _contractAddress;

late Credentials _credentials;

late DeployedContract _contract;

late ContractFunction _yourName;

late ContractFunction _setName;

late String deployedName;
```

1. **La variable _client** sera utilisée pour établir une connexion au nœud RPC Ethereum à l'aide de WebSocket.
2. **La variable isLoading** sera utilisée pour vérifier l'état du contrat.
3. **La variable _abiCode** sera utilisée pour lire l'ABI du contrat.
4. **La variable _contractAddress** sera utilisée pour stocker l'adresse du contrat intelligent déployé.
5. **La variable _credentials** stockera les identifiants du déployeur de contrat intelligent.
6. **La variable _contract** sera utilisée pour indiquer à Web3dart où notre contrat est déclaré.
7. **Les variables _yourName** et **_setName** seront utilisées pour stocker les fonctions déclarées dans notre contrat intelligent HelloWorld.sol.
8. **deploymentName** contiendra le nom provenant du contrat intelligent.

11. Fonctions

- Après avoir déclaré les variables ci-dessus, déclarez les fonctions suivantes ci-dessous :

```
ContractLinking() {
  initialSetup();
 }
 initialSetup() async {

  // establish a connection to the ethereum rpc node. The socketConnector
  // property allows more efficient event streams over websocket instead of
  // http-polls. However, the socketConnector property is experimental.
  _client = Web3Client(_rpcUrl, Client(), socketConnector: () {
    return IOWebSocketChannel.connect(_wsUrl).cast<String>();
  });
  await getAbi();
  await getCredentials();
  await getDeployedContract();
 }
 Future<void> getAbi() async {

  // Reading the contract abi
  String abiStringFile =
      await rootBundle.loadString("src/artifacts/HelloWorld.json");
  var jsonAbi = jsonDecode(abiStringFile);
  _abiCode = jsonEncode(jsonAbi["abi"]);
  _contractAddress =
      EthereumAddress.fromHex(jsonAbi["networks"]["5777"]["address"]);
 }
 Future<void> getCredentials() async {
  _credentials = await _client.credentialsFromPrivateKey(_privateKey);
 }
 Future<void> getDeployedContract() async {

  // Telling Web3dart where our contract is declared.
  _contract = DeployedContract(
```

```
      ContractAbi.fromJson(_abiCode, "HelloWorld"), _contractAddress);
  // Extracting the functions, declared in contract.
  _yourName = _contract.function("yourName");
  _setName = _contract.function("setName");
  getName();
}
getName() async {

  // Getting the current name declared in the smart contract.
  var currentName = await _client
      .call(contract: _contract, function: _yourName, params: []);
  deployedName = currentName[0];
  isLoading = false;
  notifyListeners();
}
setName(String nameToSet) async {

  // Setting the name to nameToSet(name defined by user)
  isLoading = true;
  notifyListeners();
  await _client.sendTransaction(
      _credentials,
      Transaction.callContract(
          contract: _contract, function: _setName, parameters: [nameToSet]));
  getName();
}
```

12. Création d'une interface utilisateur pour interagir avec le contrat intelligent

1. Créez un nouveau fichier nommé ***helloUI.dart*** dans le répertoire **lib/ .**
2. Ajoutez le contenu suivant au fichier :

```
import 'package:flutter/material.dart';
import 'package:geeks_for_geeks/contract_linking.dart';
```

```dart
import 'package:provider/provider.dart';

class HelloUI extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // Getting the value and object or contract_linking
    var contractLink = Provider.of<ContractLinking>(context);

    TextEditingController yourNameController = TextEditingController();

    return Scaffold(
      appBar: AppBar(
        title: Text("Hello World !"),
        centerTitle: true,
      ),
      body: Container(
        padding: EdgeInsets.symmetric(horizontal: 20),
        child: Center(
          child: contractLink.isLoading
              ? CircularProgressIndicator()
              : SingleChildScrollView(
                  child: Form(
                    child: Column(
                      children: [
                        Row(
                          mainAxisAlignment: MainAxisAlignment.center,
                          children: [
                            Text(
                              "Hello ",
                              style: TextStyle(
                                  fontWeight: FontWeight.bold, fontSize: 52),
                            ),
```

```
                Text(
                  contractLink.deployedName,
                  style: TextStyle(
                      fontWeight: FontWeight.bold,
                      fontSize: 52,
                      color: Colors.tealAccent),
                ),
              ],
            ),
            Padding(
              padding: EdgeInsets.only(top: 29),
              child: TextFormField(
                controller: yourNameController,
                decoration: InputDecoration(
                    border: OutlineInputBorder(),
                    labelText: "Your Name",
                    hintText: "What is your name ?",
                    icon: Icon(Icons.drive_file_rename_outline)),
              ),
            ),
            Padding(
              padding: EdgeInsets.only(top: 30),
              child: ElevatedButton(
                child: Text(
                  'Set Name',
                  style: TextStyle(fontSize: 30),
                ),
                style: ElevatedButton.styleFrom(
                  backgroundColor: Colors.green,
                ),
                onPressed: () {
                  contractLink.setName(yourNameController.text);
```

```
                        yourNameController.clear();
                      },
                    ),
                  )
                ],
              ),
            ),
          ),
        ),
      ),
    );
  }
}
```

- Mettez à jour le ***fichier main.dart*** comme suit :

```
import 'package:flutter/material.dart';
import 'package:geeks_for_geeks/contract_linking.dart';
import 'package:geeks_for_geeks/helloUI.dart';
import 'package:provider/provider.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // Inserting Provider as a parent of HelloUI()
    return ChangeNotifierProvider<ContractLinking>(
      create: (_) => ContractLinking(),
      child: MaterialApp(
        title: "Hello World",
        theme: ThemeData(
```

```
          brightness: Brightness.dark,
          primaryColor: Colors.cyan[400],
          hintColor: Colors.deepOrange[200]),
      home: HelloUI(),
    ),
  );
 }
}
```

14. Interagir avec l'application décentralisée complète

- Nous sommes maintenant prêts à utiliser notre application décentralisée !
- Il suffit d'exécuter le projet Flutter.

Hello World Dapp

Comme vous pouvez le constater, le message « Hello **Unknown »** dans l'interface utilisateur provient en réalité du contrat intelligent, variable **yourName** .

Lorsque vous saisissez votre nom dans le **champ TextFormField et appuyez sur le bouton surélevé** `Définir le nom` , la fonction ***setName* du fichier contract_linking.dart** sera appelée, ce qui appellera directement la fonction **setName** de notre contrat intelligent (HelloWorld.sol).

Hello World Dapp

Hello World Dapp

Hello World Dapp