Giacomo Galletti giacomo.galletti2@studio.unibo.it mat: 0000915203

BOZZA

REPOSITORY: https://github.com/GiacomoGalletti/DappMobileCrowdSensing

GLOSSARIO:

- **crowdsourcer** : soggetto che costruisce una campagna di raccolta dati secondo determinate caratteristiche e la sottopone alla rete.
  L'obiettivo del crowdsourcer è quello di ottenere dei dati secondo modalità disponibili nell'applicazione e in base al luogo geografico inserito in fase di apertura della "Campagna".

- **worker** : soggetto che volontariamente decide di partecipare ad una campagna adempiendo ad un task fornendo il dato richiesto con lo scopo di ricevere la ricompensa assegnata per quella campagna.

- **verifier** : soggetti che partecipano alla rete costruendo il blocco della catena, si accordano verificano l'attività valutando i risultati con lo scopo di percepire una ricompensa (per validators NON si intende miners)

- **campaign** : richiesta di dati sensing da parte di un crowdsourcer, nel sistema corrisponde a una creazione di uno smart contract
  Per campagna si intende una richiesta di una qualsivoglia tipologia di dati da raccogliere in una circoscritta area geografica.

- **task** : singolo contributo di un worker all'interno di una campagna, nel sistema corrisponde ad una put dell'hash nella blockchain e del corrispettivo salvataggio del dato su IPFS

Overview:

Il progetto consiste in un'applicazione de-centralizzata per un proof of concept di Mobile Crowdsensing.
L'applicazione necessita di avere l'app wallet Metamask installato sul dispositivo e impostato con un account poiché le transazioni verranno eseguite passando tramite il wallet.

Tutti i soggetti sono considerati nel modello utenti (umani) che utilizzano la medesima applicazione per svolgere le loro attività.

Una volta creata la campagna i workers possono visualizzarla e iniziare a raccogliere i dati richiesti. Il sistema verifica i requisiti geografici della campagna, ovvero se il dato proviene da un'area geografica interessata alla specifica campagna, e se approvato il dato viene caricato su IPFS ottenendo l'hash di indirizzo. Lo smart contract della campagna memorizzerà tale hash per essere raggiunto dai verifiers i quali hanno il compito di aprire i file caricati e di verificarne la validità.
Una volta che il crowdsourcer decide di chiudere la campagna avviene la distribuzione dei reward.

Tecnologie utilizzate:

- Linguaggio Dart / Framework Flutter
- backend composto da contracts in Solidity
- storage utilizza ipfs fornito tramite le API fornite da Infura

Use case:
L'applicazione al momento implementa due casi d'uso:

1. l'ottenere foto provenienti da una determinata area geografica
2. ottenere dati di rilevazione della luminosità ambientale tramite il sensore di luminosità

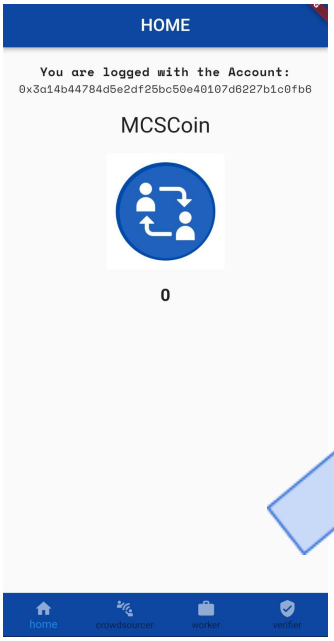
flow dello di un caso d'uso:

1. il crowdsourcer crea una campagna la quale è definita dalla definizione dei dati: titolo, posizione geografica (latitudine, longitudine), tipologia della campagna, reward per i partecipanti
2. una volta creata è possibile vedere la campagna nell'elenco delle campagne aperte presente nella sezione dedicata al worker ed al verifier
3. il worker seleziona la campagna e in base alla tipologia viene mostrata la schermata di pertinente per la raccolta dati
4. il worker esegue la rilevazione e scrive sulla blockchain
5. una volta eseguito l'upload il verifier può vedere l'elenco dei file caricati dalla sua sezione e selezionando la campagna contestuale. una volta selezionato il dato una schermata permette al verifier di visualizzare il dato e di verificarlo
6. il crowdsourcer può ora chiudere la campagna attiva e andando nella sezione apposita delle campagne chiuse può visualizzare i dati rilevati dagli utenti

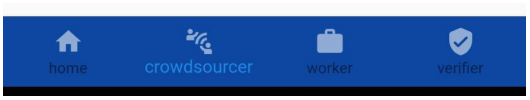
Bottom Navigation bar:

la barra di navigazione conduce nelle diverse sezioni della app che consentono di operare ognuna come un membro della rete precedentemente descritto. alla selezione della suddetta sezione divengono visibili i sotto-menù dedicati al ruolo scelto.

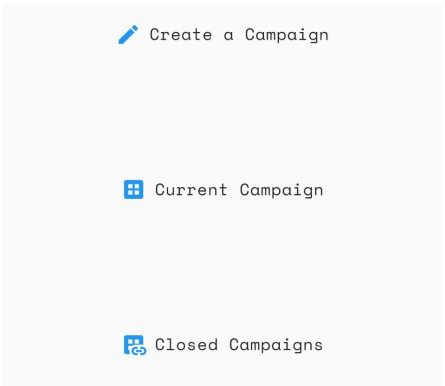
Homepage



la home page presenta un semplice wallet dove viene mostrato il quantitativo di coin posseduto dall'indirizzo che ha effettuato l'accesso.

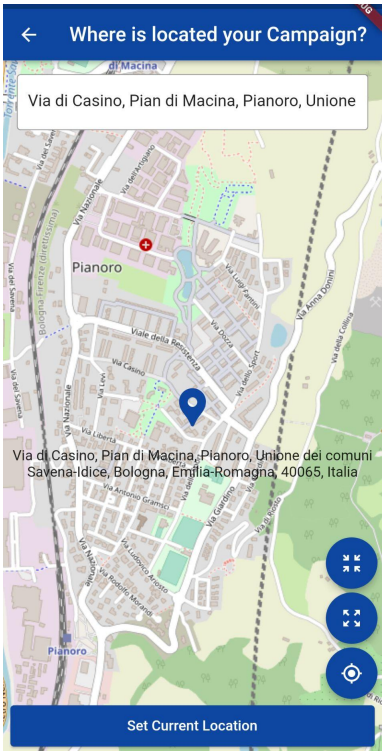


Crowdsourcer



presenta un sub-menu dove l'utente viene riportato nella specifica schermata:

- ***create a campaign***: sezione dove compilando il form è possibile eseguire una transazione di creazione per una campagna



- ***current campaign***: sezione dove è possibile visualizzare lo stato di avanzamento della raccolta dei dati da parte dei partecipanti della rete rispetto alla propria campagna attiva

- ***closed campaigns***: sezione dove è possibile visualizzare un elenco delle campagne chiuse dell'account che ha eseguito l'accesso

Worker

presenta una prima splash screen dove avviene la chiamata per recuperare le informazioni delle campagne aperte per poi portare a un elenco verticale composto da cards, ognuna delle quali riportanti le informazioni utili della campagna che rappresentano.
Alla selezione viene di una delle suddette, l'utente viene rimandato alla pagina di raccolta del dato che viene descritta qui di seguito:

- **photo** :
  in questa schermata sono presenti 2 pulsanti:
  - **take photos** :

Passa a una view di della camera dove è possibile vedere il contatore delle foto confermate finora, i settaggi per le modalità di flash e il tap to focus
il pulsante di scatto posto in basso fa scattare la fotografia per poi mostrarla subito in un widget creato a runtime dove è possibile vedere la foto appena scattata e due pulsanti: uno per tenerla e uno per scartarla
una volta confermate le fotografie, uscendo dalla camera si ritorna nella schermata dove sarà comparso una preview e il pulsante per l'upload

  - **delete all photos** :
    Elimina tutte le foto che sono state raccolte
    un pulsante per l'upload compare quando sono state scattate delle fotografie che compaiono in una piccola galleria a scorrimento orizzontale immediatamente sopra

- **light** :
  in questa schermata è presente un bottone che mostra

Verifier

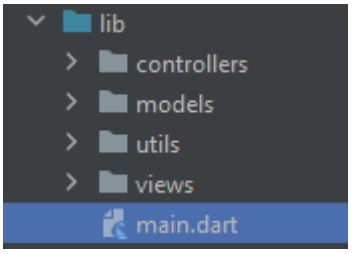
presenta una prima splash screen dove avviene la chiamata per recuperare le informazioni esattamente come nel worker. Alla selezione della campagna si viene portati all'interno della schermata contestuale alla visualizzazione del dato caricato da IPFS:

- **photo** :
  in questa schermata l'utente può vedere la galleria delle immagini che sono presenti nella rilevazione selezionata, e tramite i due pulsanti in basso può stabilire se il dato è valido o meno.
- **light** :
  In questa schermata l'utente può vedere la rilevazione della luminosità ed eventualmente attivare il suo sensore per confrontare la rilevazione. tramite i due pulsanti in basso può stabilire se il dato è valido o meno.

Implementazione:

Struttura del progetto:

l'applicazione utilizza il pattern MVC separando i widget dalla logica di ricezione dei dati che interrogano la chain.
All'interno delle classi view risiedono i componenti visuali e il widget Scaffold principale dove il controller andrà a disegnare i widget figli. Le chiamate di recupero e di upload alla blockchain e le chiamate http verso Infura risiedono all'interno delle classi Model.
all'interno della cartella /utils sono invece collocati i riferimenti a stili, font e widget utilizzati a livello globale.



Qui è possibile anche trovare due factory utilizzate come controller per generare le View appropriate in base al tipo di campagna che richiama la classe.
main.dart rappresenta l'entry point dell'applicazione dove vengono definiti i routing

Navigazione:

la navigazione avviene tramite un sistema di routing dichiarativo dove il componente Navigator mappa ogni route alla View da renderizzare. Essa è contenuta nel file main e dichiara come prima route l'entry point dell'applicazione, ovvero la home.
Dall'interno della home non si utilizza il routing ma si rimane all'interno della stessa View per renderizzare all'interno ciò che selezioniamo senza inserire nulla nello stack di Navigator.

Connessione al wallet (Metamask):
la connessione al wallet avviene grazie a un protocollo per web3
https://docs.walletconnect.com/2.0/
che consente di sfruttare API che utilizzando un'autenticazione a chiave pubblica consentono una comunicazione sicura dei dati di un wallet mediante deep link.

Tale connessione non fornisce però la chiave privata per firmare le transizioni, ma consente di mandare messaggi al wallet che vengono sfruttati dalla libreria per richiedere transazioni.

Mappa e Geolocalizzazione:
Durante la creazione di un contratto occorre inserire una posizione per identificare il "centro" in termini di latitudine e longitudine da cui bisogna essere per poter scrivere sulla blockchain.
Per questo servizio si è optato per OpenStreetMap:
https://wiki.openstreetmap.org/wiki/API che tramite le sue API offre la possibilità di ritrovare luoghi eseguendo query utilizzando il nome di punti di interesse di vario genere, che siano città, vie oppure punti di interesse.

Nella classe SearchPlacesModel sono esposti i metodi:

- *updateLocalPosition* : metodo che consente di ottenere la posizione geografica del dispositivo mediante latitudine e longitudine
- *getReadebleLocationFromLatLng* : metodo che consente di ricavare i metadati da una posizione geografica, tramite una richiesta all'API di OpenStreetMap, in questa maniera dalle geodetiche si può ricavare il nome della via o del punto di interesse associato.

Transazioni e Query su blockchain:
Per interrogare la blockchain si ha predisposto due funzioni distinte in query e transaction, nelle query non vi è una scrittura da eseguire e pertanto non occorre aprire una transazione mediata dal wallet, quindi quello che viene eseguito è la chiamata all'interfaccia ABI dello smart contract mettendo in input il nome del contratto e del metodo che si desidera chiamare.
Per le transazioni, occorre che vengano firmate con la chiave privata a cui è associato un sufficiente credito in termini di cripto per coprire i costi del gas delle operazioni di scrittura, pertanto passano attraverso al wallet chiamando un launcher che apre l'applicazione di Metamask dopo averle mandato la richiesta della transazione che verrà visualizzata e riepilogata come funzionamento del wallet.

Tali richieste vengono gestite all'interno della classe SmartContractModel che necessita in costruzione dei dati del Contratto che vogliamo caricare ovvero:

- *contractAddress* : indirizzo dove è il contratto è deployato

- *abiName* : nome del file .abi
- *abiFileRoot* : posizione del file .abi nel filesystem
- *provider* : istanza di EthereumWalletConnectProvider che espone i metodi per calcolare una firma Ethereum partendo da un indirizzo e dal messaggio. Viene fornito dalla classe sessionModel mediante il metodo getProvider

espone i seguenti metodi :
- *loadContract* : metodo che prendendo come argomento il nome del contratto carica l'abi del contratto in argomento
- *queryTransaction* : metodo per la creazione di un messaggio di una transazione
- *queryCall* : metodo per la creazione di un messaggio di una call (solo lettura)

Memorizzazione su IPFS:
le chiamate di get e put vengono eseguite all'interno del file *upload_ipfs_model.dart* qui vengono esposti i metodi seguenti:

- *uploadIPFS* : metodo per fare upload di un singolo file ritornando il suo hash
- *uploadMultipleFileIPFS* : metodo per fare upload di una lista di file
- *getOnlyHashIPFS* : metodo per pre-calcolare l'hash risultante del caricamento senza fare effettivamente il post
- *getOnlyHashIPFSDirectory* : metodo per pre-calcolare l'hash risultante del caricamento senza fare effettivamente il post ma per una cartella ed i file che contiene
- *downloadItemIPFS* : get di un oggetto presente su ipfs
- *downloadDirectoryIPFS* : get di una cartella presente su ipfs con il suo contenuto

è necessario prima di caricare il file su ipfs, verificare che sia avvenuta la transazione su blockchain, nella quale è contenuto anche l'hash di destinazione di ipfs. Grazie al metodo getOnlyHashIPFS possiamo pre calcolare l'hash per fare la chiamata della transazione, eseguendo l'effettivo post solo dopo che abbiamo verificato il buon fine della transazione

I file scaricati da ipfs finiscono in una cartella temporanea che cancella tutti i file contenuti ogni volta che vengono scaricati (ovvero quando viene creato un widget che li visualizza) Al contrario di come viene riportato nella documentazione i file vengono sempre scaricati come tar, dunque andranno estratti prima di essere utilizzati dai widget in visualizzazione. allo stato attuale su IPFS vengono memorizzati:
- file txt : per memorizzare il record del sensore di luminosità, il file verrà poi direttamente caricato su ipfs
- file jpg: per le fotografie; ogni utente può caricare una o più fotografie per ogni task. Ogni task verrà poi caricata su ipfs all'interno di una cartella che conterrà ogni foto che la compone.

Smart Contracts:

Sistema di ricompense:

Deploy e Strumenti per Testing:
L'applicazione al momento è distribuita su testnet Goerli https://goerli.net/

basata su Etherium mediante il consenso proof-of-authority, il quale risulta più robusto del consenso proof-of-stake ma altrettanto performante
Gli strumenti utilizzati per deploy e debug sono stati:

- HardHat : https://hardhat.org/
- Remix : https://remix.ethereum.org/

Problemi e Future implementazioni:
purtroppo al momento come riferimento ai contratti openzeppelin https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/structs/EnumerableMap.sol
non esiste ancora la possibilità di creare *Enumerable Maps address → address* e quindi si è dovuta mantenere molto lavoro da eseguire on chain ciclando array usati come Look Up Table delle chiavi utilizzate nelle mappe activeCampaigns e colosedCampaigns: quindi una struttura Mappa + Array per conservare le chiavi.

Questo comporta un rallentamento delle prestazioni dovuto al controllo in fase di creazione della campagna, se il sender della richiesta ha già all'attivo una campagna aperta (dovendo dunque andare in ricerca all'interno della LUT), operazione che sarebbe risultata O(1) utilizzando una mappa.

Inoltre occorre ciclare la LUT anche in fase di chiusura della campagna per eseguire il burn dell'item che prima conteneva il riferimento della campagna aperta.

Una possibile soluzione si avrebbe nell'introduzione di un Oracolo all'interno dello schema che si preoccupi di eseguire i cicli effettuando solo le richieste delle informazioni alla chain.