Mobile App: http://18.185.160.67/

API (see API endpoints!): http://3.120.6.183:5000

Github Repo:
https://github.com/Stanimir-Ivanov/sbhack-team45

MobiliPay App & API, Bchainz (sbhack-team45)

README file

Value proposition

Do you feel you have a thousand different accounts for mobility apps? 😱
Do you want new ways of using crypto assets you bravely HODLed during crypto winter? 🚀

With MobiliPay you only care about ONE account for all your mobility apps and can pay seamlessly for all your travels with your crypto assets 🙌. This gives you the freedom and independence you deserve – your personal data is managed in one place & no more expensive credit cards.

In addition, MobiliPay empowers individuals to earn ether for sharing their mobility data or performing rewardable activities defined by a transport partner - such as changing driving behavior towards more ecological transport methods.

For this hackathon, we used Ether as the payment currency as it was the easiest solution to implement. In the future, however, we can add a Swiss Franc stablecoin solution like XCHF, Rockz or stablecoin solutions still to be released (e.g. SIX), which can make the service more attractive to a wider range of consumers.

Benefits for mobility consumers:

- One account for many mobility solution apps
- Using crypto assets to pay for mobility services
- No expensive credit card needed anymore
- Democratizing online access to mobility services (not everybody gets a credit card)

Benefits for mobility providers:

- Lower barriers for users to use service
- Offering customers more innovative options to pay for services

- Less expensive payment method for mobility provider (high fees from credit card companies)
- Mobility providers have pressure to act (see introduction of Facebook's Libra coin)

MobiliPay's revenue stream consists of small fees taken from each ticket payment. s

How does it work?

Mobility service providers:

1) Technical integration

We offer mobility service providers the respective integration to include our identification and payment services into their own application (including all API endpoints). In the future, MobiliPay can be integrated with a blockchain identity platform such as eID in order to allow mobility providers to reliably authenticate users (think of car sharing requiring a valid driver's licence).

2) Wallet onboarding and management

We help mobility service providers with the creation of their Ethereum wallet and respective integration for exchanging ether into their home currency. Mobility service providers can call a withdrawal function to withdraw the funds that they have accumulated for their sold services using the MobiliPay solution as payment method, or alternatively, they can set up an automatic withdrawal mechanism to get their funds every day, month, or year.

Use of mobility services:

User has downloaded and installed the MobiliPay App:

1) Sign up for MobiliPay

In the MobiliPay application, mobility service users register with their personal data (name, address, date of birth, e-mail, …) one time and onboard their Ethereum wallet (i.e. they indicate the public key of the Ethereum address from which the funds will come from). New cryptocurrencies will be added on an ongoing basis. This new account is required to log into the mobility provider's apps without having to undergo a new registration process.

2) Login – MobiliPay App

Users log into the MobiliPay app with their newly created account where they can deposit/withdraw funds and look at their past transactions. Buying the ticket is still done through the mobility provider's own app and the identification and payment process of MobiliPay is integrated in each of these apps.

3) Top-Up / Deposit funds

After the login, the user can deposit an arbitrary amount of Ether that they want to use for paying their mobility services. This is done by calling a payable function which sends ether

from the user's Ethereum wallet to the contract where the user's balance is saved. There are multiple ways for the user to make a transfer of Ether (MetaMask, crypto exchange, …).

4) Confirm balance

The deposit transaction is confirmed and the user can view the current balance of ether in the dashboard/deposit page.

The user switches to one of the mobility provider's apps (e.g. ZVV, Publibike or Mobility):

5) Registering with a mobility provider app

The user can use the "Login with MobiliPay" function integrated in the respective mobility provider app. Therefore, no new registration is required and the data from MobiliPay can be used for the registration. MobiliPay is automatically set as a default payment method.

6) Buy tickets

When buying a ticket or paying for a mobility service the consumer confirms the purchase and pays for it seamlessly. In the backend, a MobiliPay transaction is sent to the blockchain which updates the user and mobility provider's balances without the need of an ether transaction. A small fee is accrued to MobiliPay for each transaction.

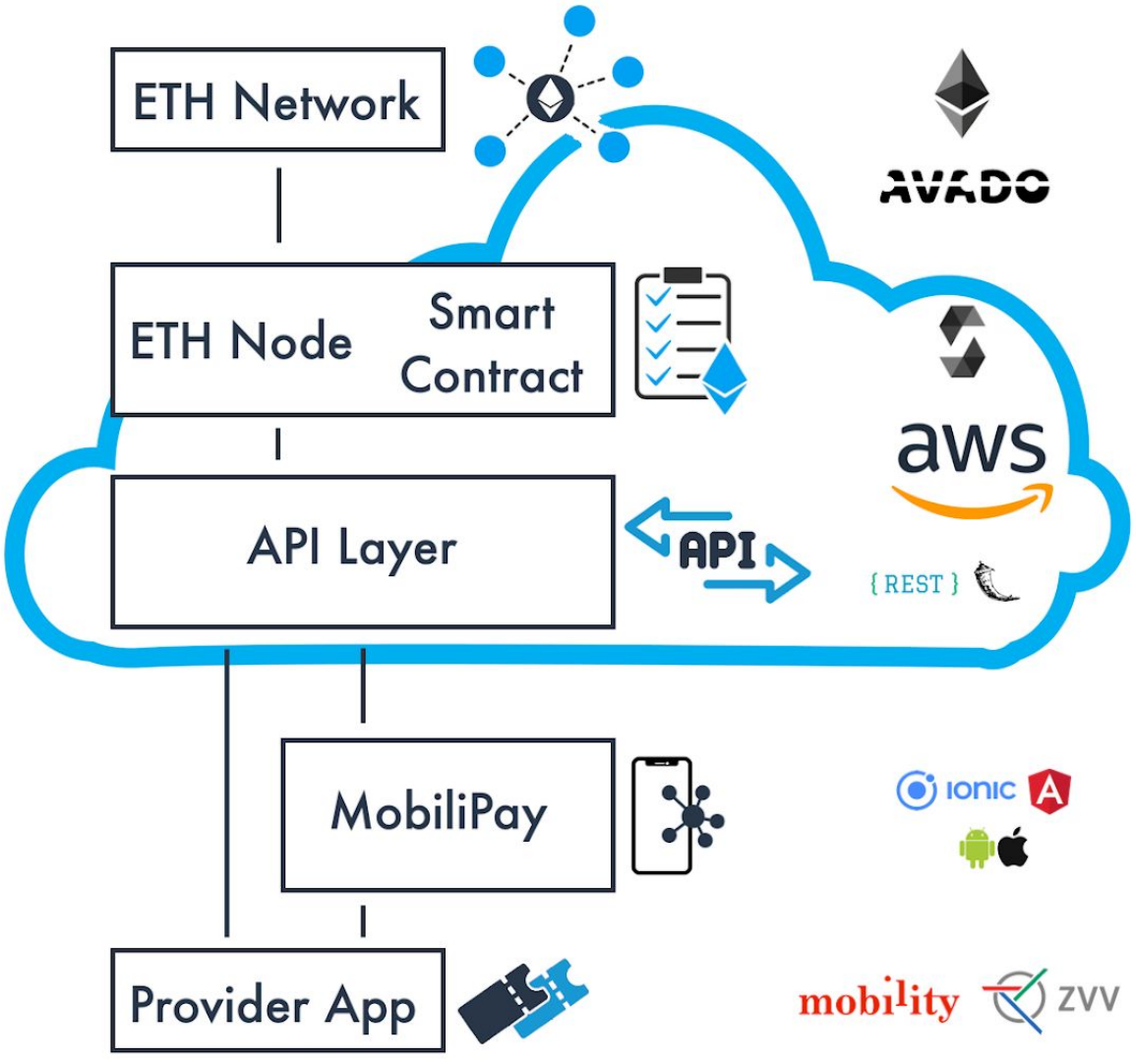
7) User gets a receipt from the mobility provider

Ticketing and invoicing is still done through the mobility provider.

8) Transaction and balance overview

The user can switch back to the MobiliPay App to check their new balance and transaction history.

Architecture Overview



Smart contract

The payment system is handled by an Ethereum smart contract called the PaymentManager. It holds a list of registered users and providers along with some auxiliary information such as provider ticket costs. Users top up their accounts in the contract by sending ether to the payable `userTopUp()` function and pay for trips with a provider by `userPayForTrip(address providerAddress)`. The contract keeps track of user and accrued provider balances internally and allows for their withdrawal. This saves on transaction costs because transport payments are only accounted for internally.

Overview of smart contract functions:

- Add user and provider addresses to the contract. The user registers only with their public address. The provider needs to register their name and ticket cost. Currently, we're modelling ticket costs by only a single ticket option but multiple tickets or cost behaviors (such as pay-as-you-go) can easily be added in the future.
    - `function providerSignUp(uint256 cost, string memory name))`
    - `function userSignup()`
- Get user and provider balances/provider costs returns the balance/cost corresponding to the account registered at the caller address..
    - `function getUserBalance()`
    - `function getProviderBalance()`
    - `function getProviderCost()`
- Payable function that allows users to deposit ether to the account linked to their public key
    - `function userTopUp()`
- Users pay for a ticket by calling this function and supplying it with the public address of the provider with which they travelled. The costs are accounted for internally by subtracting the provider's cost from the user's account and adding it to the provider's account. In this way, we minimize the need for transactions.
    - `function userPayForTrip(address providerAddress)`
- Providers can change their own costs by calling this function
    - `function setProviderCost(uint256 cost)`
- Users and service providers can withdraw their funds from the contract
    - `function userWithdraw(uint256 value)`
    - `function providerWithdraw(uint256 value)`

API - Details

We expose an API that service providers can use in order to integrate our service in their applications. This is also the API that we use in our own frontend application.

Currently, the available endpoints are as follows. To make it readable, we will consider the ip address *"10.10.10.10"*. Replace this by your actual IP address. Also change your port number

`http://10.10.10.10:5000/api/user_signup`

`http://10.10.10.10:5000/api/provider_signup?cost=yourAmount&name="provider_name"`

`http://10.10.10.10:5000/api/topup?amount=yourAmount`

`http://10.10.10.10:5000/api/withdrawProvider?amount=yourAmount`

`http://10.10.10.10:5000/api/withdrawUser?amount=yourAmount`

`http://10.10.10.10:5000/api/pay?to=providerAddress`

`http://10.10.10.10:5000/api/getBalanceUser`

`http://10.10.10.10:5000/api/getBalanceProvider`

`http://10.10.10.10:5000/api/getCostProvider`

`http://10.10.10.10:5000/api/setCostProvider?cost=yourAmount`

NOTE: this backend service - and its provided endpoints - are not meant to be called individually, as they will probably not make sense if you don't configure them properly (e.g. proper credentials). It is intended as a means of communication with the blockchain for the MobiliPay app directly or for integration in provider applications.

Please also pay attention to the type of the arguments (quotation mark or not). E.g.:

`http://10.10.10.10:5000/api/withdrawUser?amount=1.23` **will work**

`http://10.10.10.10:5000/api/withdrawUser?amount="1.23"` **will <u>not</u> work**

Backend deployment

The backend should run on a dedicated server. Clone the repo, and execute the following commands. You need to have python 3.6 at minimum for it to work, and to have pip installed.

```
cd backend/src/
python3.6 -m venv venv
source venv/bin/activate
pip3 install flask flask-cors web3 py-solc
```

Now, you can set-up a local blockchain dev node as follow:

```
npm install -g ganache-cli
npm install -g truffle
ganache-cli -h "your_ip"
```

In a new terminal, navigate to *"/payment-manager/"*. Edit the *"truffle-config.js"* file such that under *"development"* the ip address and the port number match those of your server. Once this is done:

```
truffle migrate
```

This while compile the solidity contracts and publish them on the local node.

On both terminals, in the standard output, there are a few information which you need to note down:

- from ganache-cli:
  - The ip address on which the node is running

- A private key of your choice (this will give you access to a pre-filled wallet, as a user)

- from truffle:

  - The PaymentManager contact address

Finally, you can start up the backend service as:

```
cd /backend/src/
```

```
python3.6 main.py --provider_address="http://your_ip:your_port"
--contract_address="the_contract_address" --private_key="the_private_key"
```

If you did everything right, you will be able to access the API at your public IP address, followed by the postfix '/api/'

Frontend

Many day-to-day functions like payment are to be integrated directly into partner apps like ZVV, Mobility, etc. The MobiliPay standalone app covers additional features like checking balance, topping up and withdrawal, and viewing past transactions. The app interacts with the API layer to fetch data and execute transactions.

The mobile app frontend is based on Ionic 4 and Angular 7. The Ionic CLI was used to generate the empty template, and the CLI is also used for the various build steps. Ionic enables cross-platform mobile support for iOS and Android and allows the app to be previewed by a mobile browser or by using the mobile emulation mode in a desktop browser.

Prerequisites: Node.js / npm

- `npm install -g ionic`

In folder bchainzapp

- `npm install`
- To run locally: `ionic serve`
- To build: `ionic build --prod`
- The contents of the `www` folder can be deployed to a web server to provide a browser based preview. (Example nginx config that works with the angular router is in `config/frontend-webserver/`)

Native iOS build

- Follow the steps on https://ionicframework.com/docs/building/ios